PL-TR-
96-1133

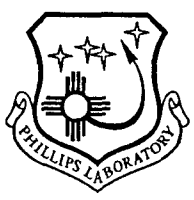
ADAPTIVE NEURAL CONTROL FOR SPACE STRUCTURE VIBRATION SUPPRESSION

Lawrence D. Davis
David C. Hyland

Harris Government Aerospace Systems Division
P. O. Box 94000
Melbourne, FL 32902

August 1996

Final Report



PHILLIPS LABORATORY
Space Technology Directorate
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

AD NUMBER

AD-B216 141

NEW LIMITATION CHANGE

TO

DISTRIBUTION STATEMENT A -
Approved for public release; Distribution unlimited.

Limitation Code: 1

FROM

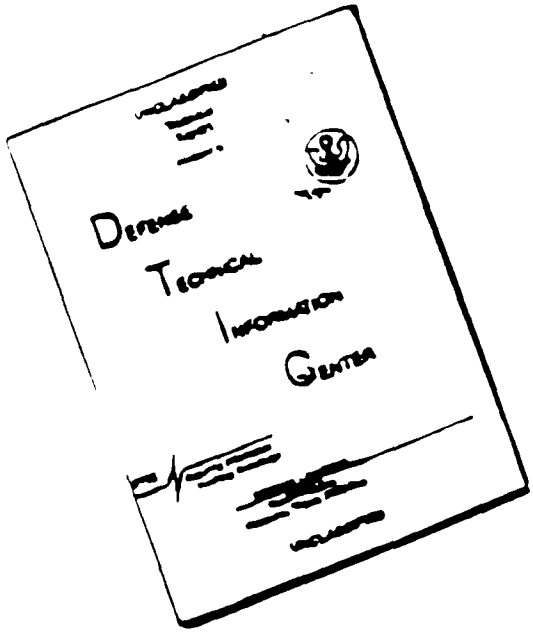
DISTRIBUTION STATEMENT -

Limitation Code:

AUTHORITY

Janet E. Mosher; Phillips Lab/CA, Kirtland AFB, N.M.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

PL-TR-96-1133

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data, does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

If you change your address, wish to be removed from this mailing list, or your organization no longer employs the addressee, please notify PL/VTS, 3550 Aberdeen Ave SE, Kirtland AFB, NM 87117-5776.

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

This report has been approved for publication.

GARY YEN
Project Manager

FOR THE COMMANDER

L. KEVIN SLIMAK, GM-15
Chief, Structures and Controls Division

CHRISTINE M. ANDERSON
Director, Space Technology
Directorate

The following notice applies to any unclassified (including originally classified and now declassified) technical reports released to "qualified U.S. contractors" under the provisions of DoD Directive 5230.25, Withholding of Unclassified Technical Data From Public Disclosure.

NOTICE TO ACCOMPANY THE DISSEMINATION OF EXPORT-CONTROLLED TECHNICAL DATA

1. Export of information contained herein, which includes, in some circumstances, release to foreign nationals within the United States, without first obtaining approval or license from the Department of State for items controlled by the International Traffic in Arms Regulations (ITAR), or the Department of Commerce for items controlled by the Export Administration Regulations (EAR), may constitute a violation of law.

2. Under 22 U.S.C. 2778 the penalty for unlawful export of items or information controlled under the ITAR is up to two years imprisonment, or a fine of $100,000, or both. Under 50 U.S.C., Appendix 2410, the penalty for unlawful export of items or information controlled under the EAR is a fine of up to $1,000,000, or five times the value of the exports, whichever is greater; or for an individual, imprisonment of up to 10 years, or a fine of up to $250,000, or both.

3. In accordance with your certification that establishes you as a "qualified U.S. Contractor", unauthorized dissemination of this information is prohibited and may result in disqualification as a qualified U.S. contractor, and may be considered in determining your eligibility for future contracts with the Department of Defense.

4. The U.S. Government assumes no liability for direct patent infringement, or contributory patent infringement or misuse of technical data.

5. The U.S. Government does not warrant the adequacy, accuracy, currency, or completeness of the technical data.

6. The U.S. Government assumes no liability for loss, damage, or injury resulting from manufacture or use for any purpose of any product, article, system, or material involving reliance upon any or all technical data furnished in response to the request for technical data.

7. If the technical data furnished by the Government will be used for commercial manufacturing or other profit potential, a license for such use may be necessary. Any payments made in support of the request for data do not include or involve any license rights.

8. A copy of this notice shall be provided with any partial or complete reproduction of these data that are provided to qualified U.S. contractors.

D E S T R U C T I O N N O T I C E

For classified documents, follow the procedures in DoD 5200.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

DRAFT SF 298

1. Report Date (dd-mm-yy) August 1996	2. Report Type Final	3. Dates covered (from... to) 09/93 to 08/96
4. Title & subtitle Adaptive Neural Control for Space Structure Vibration Suppression		5a. Contract or Grant # F29601-93-C-0147
		5b. Program Element # 62601F
6. Author(s) Lawrence D. Davis David C. Hyland		5c. Project # 8809
		5d. Task # TM
		5e. Work Unit # GY
7. Performing Organization Name & Address Harris Government Aerospace Systems Division P. O. Box 94000 Melbourne, FL 32902		8. Performing Organization Report #
9. Sponsoring/Monitoring Agency Name & Address Phillips Laboratory 3550 Aberdeen Ave SE Kirtland AFB, NM 87117-5776		10. Monitor Acronym
		11. Monitor Report # PL-TR-96-1133

12. Distribution/Availability Statement
Distribution authorized to U.S. Government agencies and their contractors only; Critical Technology; August 1996. Other requests shall be referred to AFMC/STI.

13. Supplementary Notes

14. Abstract Despite recent advances in efficiency, current methodologies for space structure control design still engage significant human resources for engineering development and maintenance. The work reported in this document is part of an effort to develop neural network based controllers capable of self-optimization, on-line adaptation and autonomous fault detection and control recovery. Focusing on vibration suppression controllers, we reviewed previous work that experimentally demonstrated the application of a new neural network architecture to feedback control and feedforward control of tonal disturbances. New developments included efficient and completely autonomous neural network feedforward control for the case of broadband disturbances.

15. Subject Terms Neural Networks, Active Control, Vibration

Security Classification of			19. Limitation of Abstract	20. # of Pages	21. Responsible Person (Name and Telephone #)
16. Report Unclassified	17. Abstract Unclassified	18. This Page Unclassified	Limited	214	Gary Yen (505) 846-8256

Table of Contents

List of Figures

List of Tables

Adaptive Neural Controls (ANC) Executive Summary

Despite recent advances in efficiency, current methodologies for space structure control design still engage significant human resources for engineering development and maintenance. For example, since fixed-gain space system controllers must be updated periodically to adjust to in-mission changes in system dynamics, this implies burdensome ground support activities.

The Adaptive Neural Controls (ANC) program is part of an effort to develop neural network based controllers capable of self-optimization, on-line adaptation and autonomous fault detection and control recovery. This development also supports the Nation's long-term space exploration objectives for which autonomous spacecraft involving self-reliant control systems are a necessity.

The ANC Program was funded in two phases. The first, Basic phase focussed on the development of efficient and completely autonomous neural network feedforward control for the case of broadband disturbances. Algorithms were developed that work with no prior modeling information about the system to be controlled and adapt to changing conditions, while minimizing or eliminating the introduction of extraneous training signals. The algorithms were demonstrated experimentally on an optical structure testbed at Harris.

The second, Amended Option 1 phase of the program demonstrated a more complex neural controller on the ASTREX test facility at the Air Force Phillips Laboratory capable of fault-tolerant adaptive control of multiple sensors and actuators. This system used 6 actuation channels of the existing ACESA struts on the ASTREX structure to simultaneously cancel three independent tonal disturbances in the 10-15 Hz band, measured at non-collocated sensors on the secondary tower of the structure. The system demonstrated impressive fault-recovery performance, maintaining good cancellation performance with successive actuators disabled, down to a minimum of two out of six. Cancellation of individual tones was between 20 and 50 dB, with over 28 dB attenuation realized RMS. The algorithm required very low computational throughput, operating at a sample rate of 1/20 Hz.

The results of the ANC program show that adaptive cancellation systems can reduce vibrations in precision structures without prior modeling data and can adapt successfully to failures in actuators or sensors, optimally reconfiguring themselves without human intervention. These capabilities should significantly reduce the expense of designing and maintaining vibration control systems for spacecraft.

The next step in the development of ANC technology should be to demonstrate broadband adaptive neural cancellation using a small flight-like processor such as the Modular Control Patch. This would give potential users confidence that the technology could be integrated into their existing or planned systems with little overhead in terms of cost, volume, weight, or power.

1. Introduction

With the advent of Adaptive Neural Control (ANC), we are on the threshold of revolutionary changes in control design and implementation. These changes will impact not only structural vibration control but a wide variety of other controls tasks as well. Just in the area of vibration control, the advances made recently in ANC translate into greater cost-effectiveness for control design, implementation and verification and a corresponding expansion of DoD and commercial applications.

By way of introduction, we first trace the historical trends in active structural controls that have stimulated and provided the basis for ANC development at Harris. We devote careful effort to this discussion in order to convey our vision of how ANC fits in with the on-going evolution of structural control technology. The historical record suggests that any technology undergoes several distinct phases in its development: 1) an *embryonic* phase in which fundamental data is gathered and the essential difficulties are identified and grappled with, (2) a *basic competence* stage in which full-scale capabilities are demonstrated but the technology is still costly and cumbersome in its implementation and not necessarily effective in its results and, finally (3) the *incisive* stage wherein not only can the technology be applied routinely with the desired success but the processes of its application are quick and inexpensive. The third, incisive stage must be reached in order for the technology to find use in a broad range of industrial and commercial applications.

The area of active structural control design for aerospace and commercial systems already display at least the first two development phases noted above. Although, at present, there are groups still working within nearly every point of the development sequence, the state of the art is now entering the third phase.

The first, embryonic phase of structural control design began in the mid 70's and extended into the early 80's. This early stage involved much theoretical effort in its attempts to apply "modern control theory" (Kalman filtering and optimal feedback control theory developed in the 60's) to vibration control problems. During this period also, many sub-scale, highly idealized experiments were performed to test the basic concepts and identify the significant roadblocks to effective performance. As a result, several difficulties with the earlier control theories were identified and efforts were made to resolve them. For example, two of the most severe challenges were the design of (1) *simplified* controllers (low order dynamic controllers for high order systems) and (2) *robust* controllers that maintain both stability and acceptable performance in the face of system modelling errors and in-mission changes in physical characteristics. In response to these challenges, a variety of reduced-order and robust control design methods were developed and the feasibility of at least a subset were demonstrated via small scale laboratory experiments. Although this first development phase produced most of the fundamental tools and illustrated their feasibility, convincing evidence was still lacking for the claimed performance benefits of the technology when applied to full scale structural control problems.

The second, basic competence, phase of development involved the competitive sorting out of a plethora of various design approaches, using test experiences on larger scale, nonidealized apparatus. A really positive stimulus was provided by the emergence of "Guest Investigator" efforts which provided several investigative groups the opportunity to conduct research on

traceable testbeds that would otherwise be beyond the technical and financial means of individual researchers. An example is the NASA Controls/Structures Integration (CSI) Guest Investigator (GI) Program, that recently completed its second phase. Both guest investigator programs and other large scale experimental efforts involved *independently refereed* testbeds—i.e., the hardware is provided and hosted and the control algorithms are actually implemented (and test data taken) by an independent Government appointed body. This arrangement provided a completely independent mechanism for unbiased, critical review of competing approaches and we restrict discussion here to independently referred test results only. Also, of course, our discussion includes only those efforts in which Harris researchers participated. This experience however, amply illustrates how the basic competence phase of technology development was successfully negotiated and how the way was opened to the third stage of development.

Harris researchers have demonstrated vibration control design on *six* independently refereed testbeds. In Figure 1-1 we show, for each testbed, the values of:

Number of Performance Degrees of Freedom

= number of generalized degrees of freedom that must be independently suppressed to address performance requirements

Number of Modes in Disturbance Band

= number of vibration modes in the disturbance band that must be attenuated by at least an order-of-magnitude in order to meet performance requirements

These two variables give a rough indication of the difficulty of the control problem. The ACES, Mini-MAST, CASES, CEM Phase 2, and CEM Phase 3 test facilities were addressed on the NASA CSI GI program in the order given. Harris was the only investigator group to participate on both phases of this program. The performance results achieved are documented in [1]. Most of the testbed experiments involved only control algorithm design with the exception of CEM Phase 3 and ACTS which also involved test demonstrations of Harris' inertial proof-mass actuators and active isolation intrastructural actuators.

The above testing experiences using large scale facilities under nonidealized conditions, helped us (and other groups) progress beyond the embryonic stage of development in two important ways. First, the early experimentation on ACES and Mini-MAST showed that the development of advanced design tools (such as Matlab toolboxes for Maximum Entropy/Optimal Control, μ-Synthesis, or Quantitative Feedback Design) is not enough for efficient (or even successful) implementation of working control systems in practice. What is needed, in addition, is a practical *methodology* for using advanced design tools that meshes algorithm design, system modeling and subsystem and component tests, into a realistic strategy for implementation. The development of such a methodology was key to our success. The second area of progress was the *streamlining* of the whole design/test/validation process to achieve validated control results with

less time and expenditure. For example, the cumbersome, time consuming process of building and refining finite-element models was replaced by the use of automated system identification methods for extracting dynamic models directly from test data. This approach fits in with on-orbit control refinement for space systems or with any kind of "retrofit" situation in which vibration control is desired for existing structural hardware. A second area of streamlining was the introduction of faster design tools that permitted design turnaround on-site.

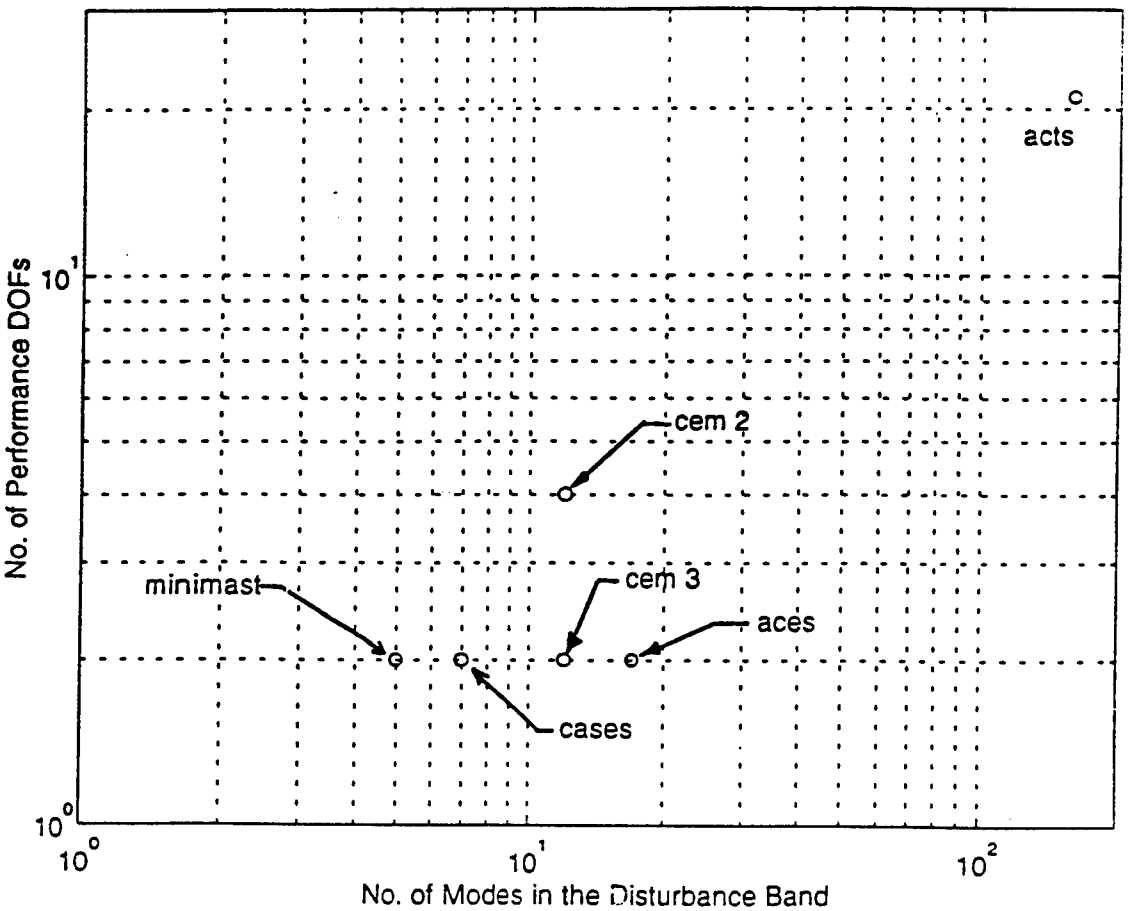


Figure 1-1: Six Independently Refereed Vibration Control Experiments have been Completed by Harris Researchers

The development of a systematic design/test methodology and its subsequent streamlining significantly improved efficiency over the course of the GI program. For example, the ACES facility at NASA/MSFC (the first testbed addressed) required a man year and three on-site test sessions (2 days each): An initial session for open-loop testing, a second to test an initial, robust controller (while collecting closed-loop ID data) and a third session to test the integrated, advanced controller. In contrast, a few years later, the CEM Phase 2 testbed, required two months of design effort and a *single* on-site test session. This result is obtained despite the fact that the CEM Phase 2 and ACES facilities were of comparable difficulty (see Figure 1-1).

The above efficiencies were essential to successfully address the most challenging structural control problem yet posed: the ACTS program.

On this program, the structural testbed (Figure 1-2) is provided by an independent host contractor. The testbed is very complex, traceable and program-directed. Disturbances at the base have flat PSDs from 10 to 200 Hz. Stringent performance requirements are levied on both line-of-sight (LOS) and aberration errors, such as defocus and primary reflector surface wavefront errors. Because of the very broadband disturbances, tens of modes contribute to LOS error and more than a hundred modes contribute to wavefront error. In all, the response of over 150 modes must be attenuated by more than 20 to 40 dB in order to meet performance specifications. The Harris effort involves not only control algorithm design but also the fabrication, integration and test of two distinct hardware complements: 24 Advanced LPACTs and 6 Active Isolator Fittings (AIFs). In all, the control design problem entails 30 inputs and 30 outputs. Our mission was to quantify, as comprehensively as possible, the complexity and cost versus performance tradeoffs inherent in the application of active structural control to this class of systems.

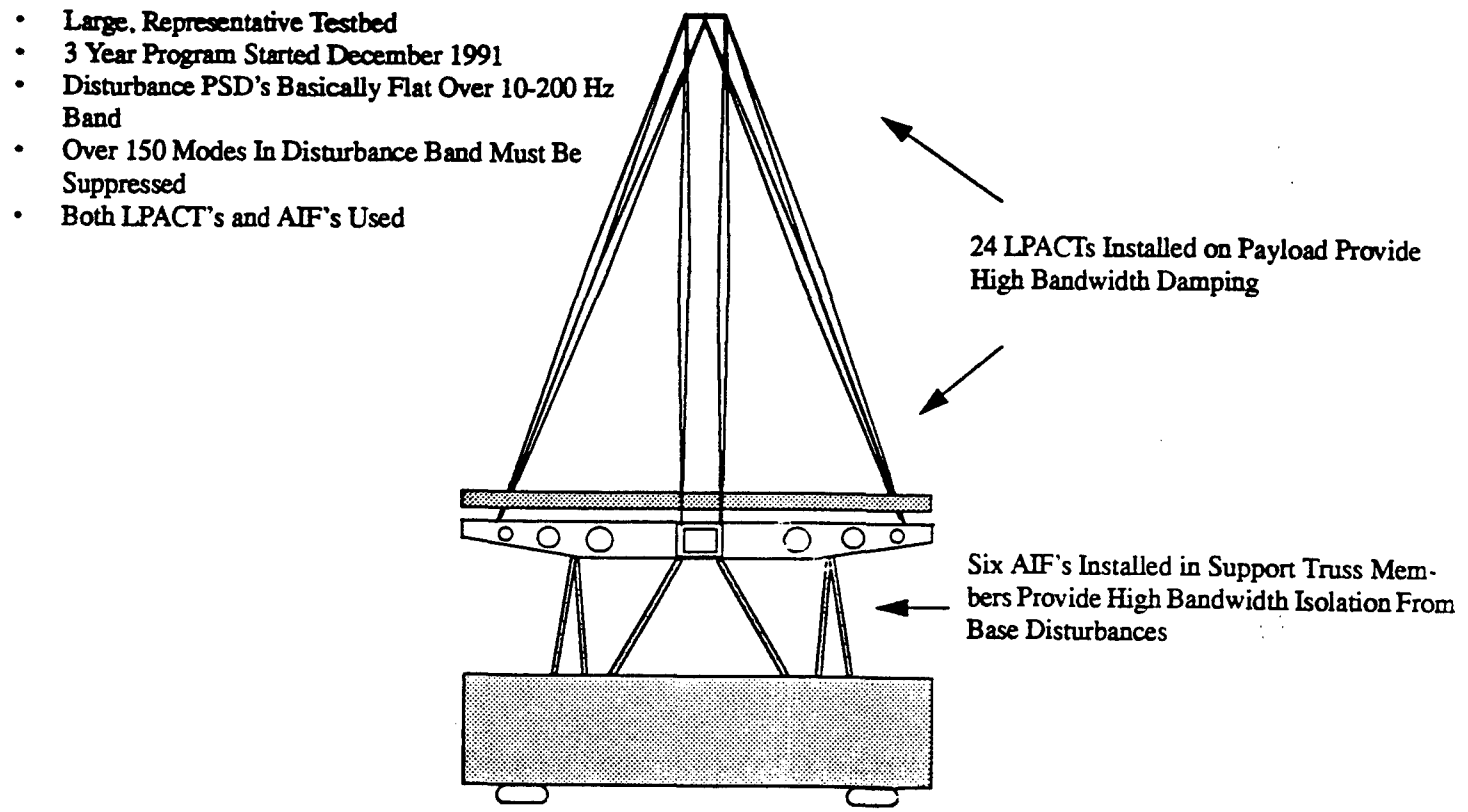


Figure 1-2: The ACTS Testbed is One of the Most Difficult and Traceable in Existence

Test data shows over 40 dB attenuation of LOS related modes. Very significant results were also achieved for aberration errors. Figure 1-3 shows open-versus closed-loop PSD test data results for defocus. Nearly 50 dB attenuation is achieved for the dominant modal peak and over 30 dB rms is observed over the 10 to 200 Hz frequency band. Finally, Figure 1-4 shows open-versus

closed-loop PSD data for wavefront error. Evident in this plot is the large number of modes contributing to wavefront error in the open-loop. Dominant modal peaks are reduced by nearly 40 dB and over 20 dB rms attenuation is obtained over the entire disturbance band.

In all of the test programs in Figure 1-1, the fixed-gain control designs invariably worked on the first trial and performance specifications for each testbed were invariably met or exceeded. Because of the effectiveness and predictability of the results, the independently refereed testing experiences did much to make the technology an area of *professional competence* backed by *reliable, effective tools*—and this is the worthy goal of the second, "basic competence" phase of technology development. Our efforts to streamline the design and test process not only reduced cost and engineering design time but also led us to the threshold of the third, "incisive" phase of development. Our results suggested the idea that if one can systematize and streamline a human-operated design/test process (such that modeling, control design and test verification are all accomplished in a single on-site session), then one ought to be able to automate the entire

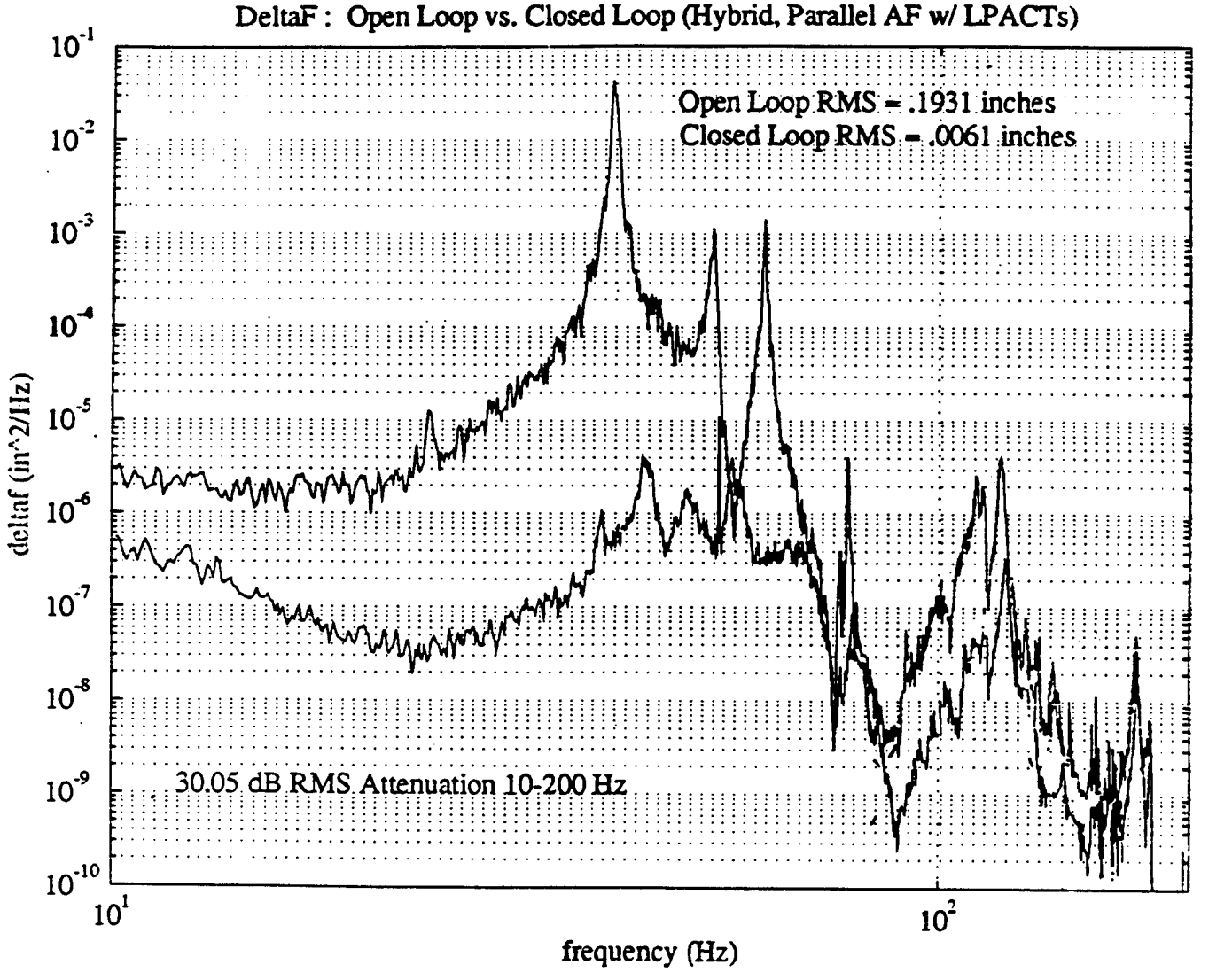


Figure 1-3: ACTS Defocus Errors are Reduced by More Than 30 dB rms Over the 10-200 Hz Disturbance Band and Peak Amplitudes are Attenuated by Nearly 50 dB

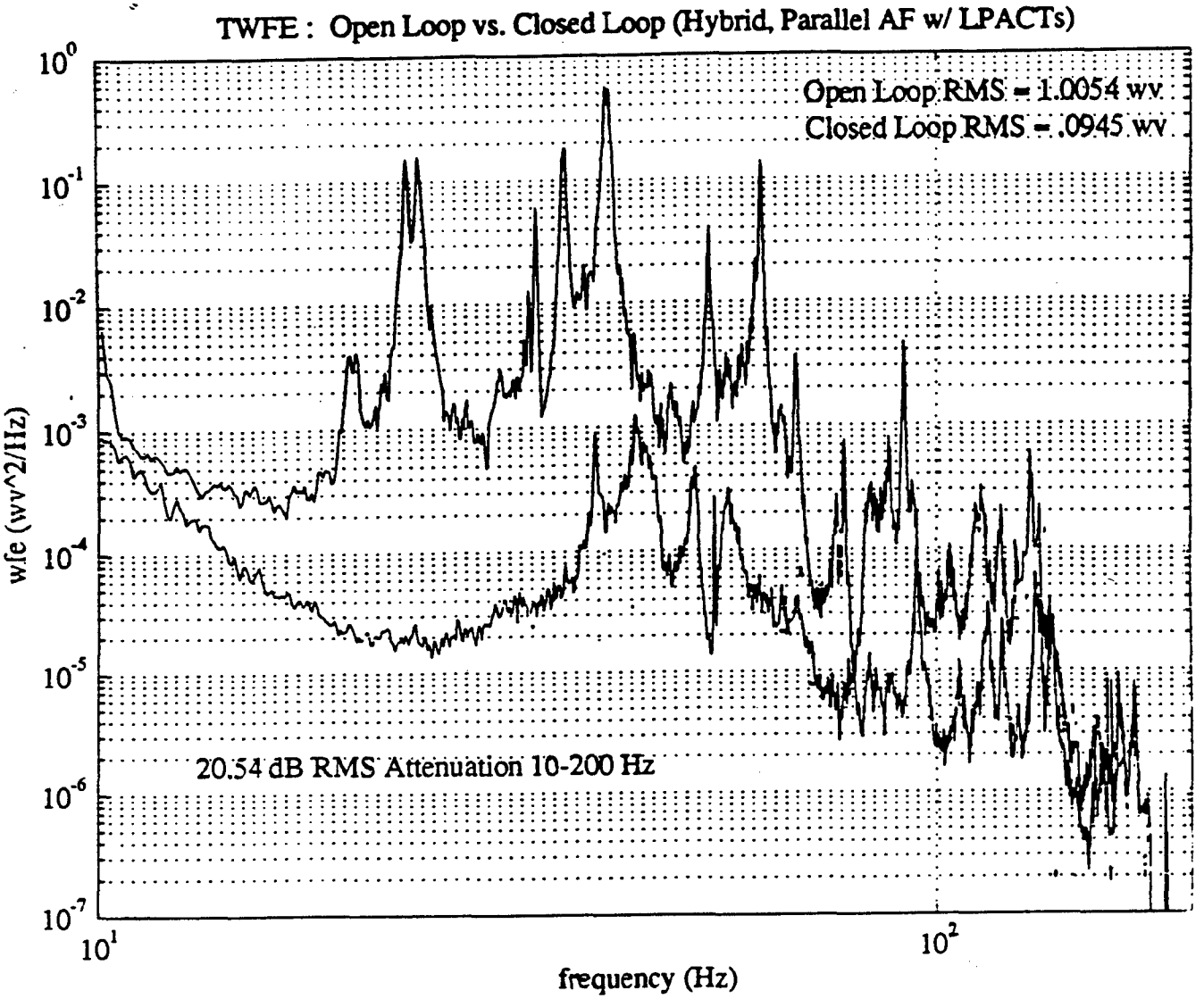


Figure 1-4: ACTS Wavefront Errors are Reduced by More than 20 dB rms Over the 10-200 Hz Disturbance Band and Peak Amplitudes are Attenuated by More than 40 dB

process. This is the first motivation for ANC. The second motivation arose because the invention of a new neural network architecture for addressing dynamic systems by Dr. D.C. Hyland in 1989 provided an algorithmic means for realizing the desired automation. Finally, over the intervening years, advances in Digital Signal Processor (DSP) technology provided ever faster and cheaper hardware capable of implementing the ANC algorithms in real time.

As mentioned, the incisive stage of technology development is attained when not only is the technology proved capable of reliable results but the implementation of the technology is rapid and relatively cheap. Total automation of the control design/test process is needed because the current methodology still engages significant human resources. Since space system control designs involving fixed-gain controllers must be updated periodically to adjust to in-mission changes in system dynamics, this implies burdensome ground support activities. Besides

reducing engineering manpower requirements, such advances in automation support the Nation's long-term space exploration objectives for which autonomous spacecraft involving a self-reliant control systems are a necessity. Such robot explorers would have to independently update control laws, detect faults and reconfigure control systems. Finally, while there are numerous commercial areas that would use or benefit from active vibration control, this will not occur until dramatic decreases in engineering development cost and dramatic improvements in system self-reliance are achieved.

2. The Adaptive Neural Control Architecture—Overview Description

For the past four years, motivated by the technology needs described above, Harris has been developing a new neural architecture to implement on-line systems identification and adaptive control for systems. Applications were originally targeted to vibration suppression in precision space structures but have since been expanded to noise and vibration control and motion control for both DoD and commercial systems. Our studies began in 1989 with the discovery by Dr. D.C. Hyland of a new neural architecture for identification and control. The new architecture, termed the Adaptive Neural Control (ANC) architecture, subsequently led to a sequence of successful demonstrations and new development efforts.

Much of the previous work in adaptive control via neural networks (see [2] for an excellent review) concentrated on highly nonlinear but low dimensional systems. In contrast, the ANC architecture concentrates on neural schemes particularly geared to problems involving high order systems exhibiting very broadband dynamics. As indicated in Figure 2-1 ANC combines tapped delay lines with "static" neurons (each neuron is a two-way device incorporating a back propagation path) to perform on-line system identification and adaptive control. The system adapts in the presence of unknown persistent plant disturbances and instrumentation noise and requires no detailed prior modeling information.

There are several key features of this architecture that have made it particularly attractive. First, although the architecture can be visualized as a neural network, the control scheme is fundamentally a massively parallel, decentralized adaptive control algorithm that need not be *implemented* literally as a collection of artificial neurons. Secondly, these "neural" algorithms feature learning capability that is distributed down to the smallest computational unit. Decentralization (distributed learning) imparts the ability to autonomously recover from hardware failures—including damage to the neural processor itself. A third key feature is that the basic neural building blocks are hierarchically organized into a set of standardized modules. Analogous to a "Lego set," modules can be combined to build an enormous variety of systems and permits complex systems to be built up from simpler components in a transparent way. Finally, modularity and parallelism yield implementation flexibility. Specialized hardware is *not* required for implementation of the Harris ANC architecture. The entire identification or control algorithm can be distributed among several parallel processors, and hardware suitable for this purpose is currently available and is being used for engineering development. This means that we can progress in orderly fashion from the use of existing ICs to (ultimately) dedicated neural ICs, thereby building our capabilities gradually and systematically.

While details of the ANC architecture are given in recent papers and reports [3]-[5], we briefly review the basic features here. The hierarchy of modular structures is shown in Figure 2-2. This hierarchy starts, at the lowest level, with tapped delay lines and neurons with intrinsic back propagation. These are the same "static" neurons that would be utilized for such applications as pattern classification and nonlinear mapping. The key to applying such neurons to dynamic system identification is to organize them into larger building blocks, the *dynamic ganglia*. A ganglion is an array of neurons designed to establish temporal ordering within the network so as to process

time histories of network signals. Ganglia are interconnected by bundles of synapses, which are sometimes constrained so that their weights form Toeplitz matrices.

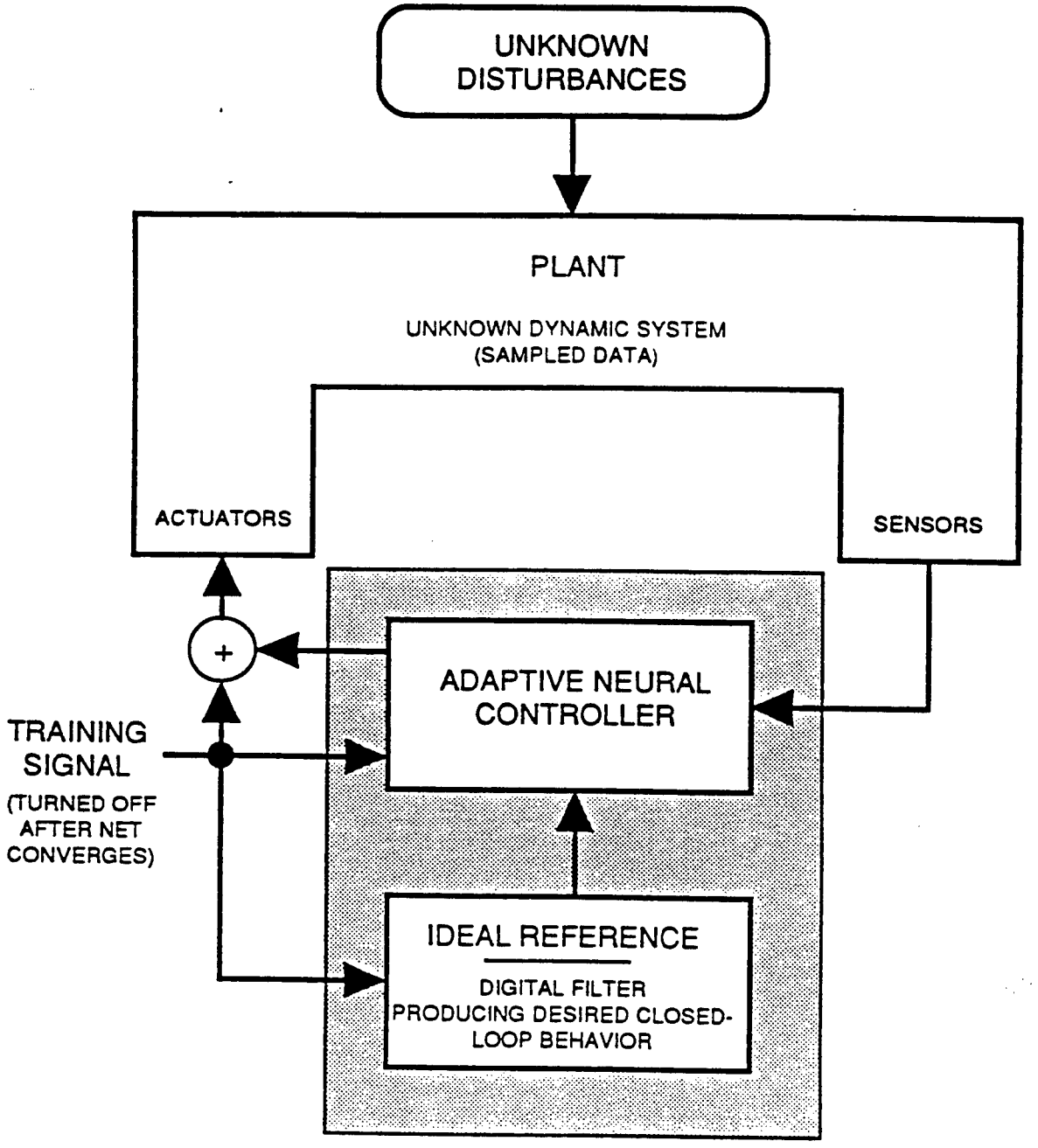


Figure 2-1: The Adaptive Neural Controller Executes Simultaneous System Identification and Adaptively Optimized Control

The next level in the hierarchy combines ganglia and their connecting synaptic bundles to form replicator units. The basic job of a replicator unit is to duplicate the output of a previously unknown sampled – data dynamic system when both replicator and system are stimulated by the

same training input. Thus the replicator is the basic module for system identification. Several types of replicator have been developed, each corresponding to a particular model form. The work in identification methods using systems observer Markov parameters [6], [7], led to discovery of a new model form for dynamic systems – the ARMarkov model, so-called

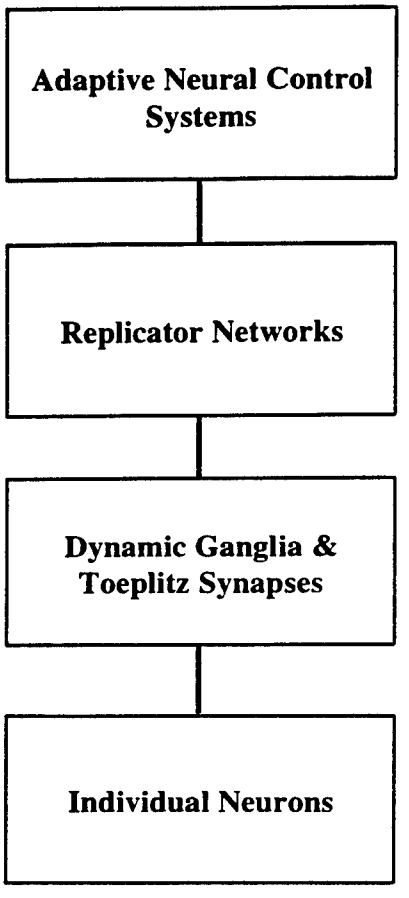


Figure 2-2: Hierarchy of Modular Nebular Structures Progressing from Basic Constituents to Higher-Level Modules

because it combines features of impulse response (Markov parameters) with ARMA (Auto Regressive Moving Average) models.

Furthermore, most of the replicator types have both time and frequency-domain (or mixed time and frequency-domain) versions. Generally, one obtains a frequency-domain version of a particular time-domain replicator by replacing the time series inputs and the time series error signals by the outputs of Finite Fourier Transform (FFT) filters having these signals as inputs. This operation is equivalent to inserting a multiplication by a unitary matrix within the replicator structure. However, this superficially trivial change allows the replicator to address frequency weighted output matching and direct frequency response identification.

Numerous analytical examples have been produced to demonstrate structural identification or control using neural replicators. Some of these involve the use of simulated input/output data and others use actual test data. Also, a number of laboratory experiments have been performed.

In some cases a MATLAB simulation was used to implement the neural algorithm, while in other cases the algorithm was implemented in real time using a DSP card. For example:

1. Figure 2-3 shows a simple beam experiment that produced excellent convergence of the adaptive model to the actual structural plant in 100 seconds.
2. The neural network system identification capability was also demonstrated on the Harris Multi-Hex Prototype Experiment (MHPE), which is a four meter Cassagrain test structure. Figure 2-4 shows the network converged to the MHPE plant in 125 seconds.
3. Using a Digital Signal Processor (DSP), an Internal Research and Development (IR&D) experiment in active acoustical noise cancellation was completed in which over 20 dB broadband attenuation was achieved.

Many of these examples involve multiple inputs and outputs and nearly all involve fairly complex structures with many modes in the frequency band of interest. Also the laboratory experiments tested the algorithm under such real-world complications as sensor noise and ambient steady-state disturbances. Summarizing this experience, we can say that reasonably complex multi-mode systems can be identified with excellent accuracy with convergence times ranging from a few minutes to a few seconds (depending on numerous factors, such as system sample rate, frequency band of interest, etc.).

Returning now to the hierarchy shown in Figure 2-2, several replicator units are combined in order to form the Adaptive Neural Control (ANC) system. An ANC performs on-line, simultaneous system identification and adaptively optimized control. The most basic ANC architecture for simultaneously replicating an unknown plant and adapting an output feedback controller so as to match the closed-loop input/output characteristics with a prescribed reference system has two parts: (1) the closed-loop modeller and (2) the control adaptor. The closed-loop modeller uses training signals and the plant sensor output to adapt the weights so that the closed-loop is replicated. After convergence, the modeller output matches the closed-loop system – in effect the modeller identifies the plant within the closed loop.

In the control adaptor, there is an internal model of the plant, copied from the plant modeller. Thus the control adaptor can, in effect, back-propagate error through the plant model to the controller output location. With its internal model of the plant, the adaptor uses the training signal, its own output and that of the reference system to adjust its weights so that the reference system is replicated.

The above form of ANC addresses the feedback type of control architecture. It is useful here to note the two quintessential architectures for adaptive control: feedforward cancellation and feedback control. Figure 2-5 illustrates these two basic disturbance suppression strategies. In feedforward cancellation, the system motion measurements (the error sensor outputs) are used only to adjust the feedforward gain, F. Once F converges, the actuators are driven by the

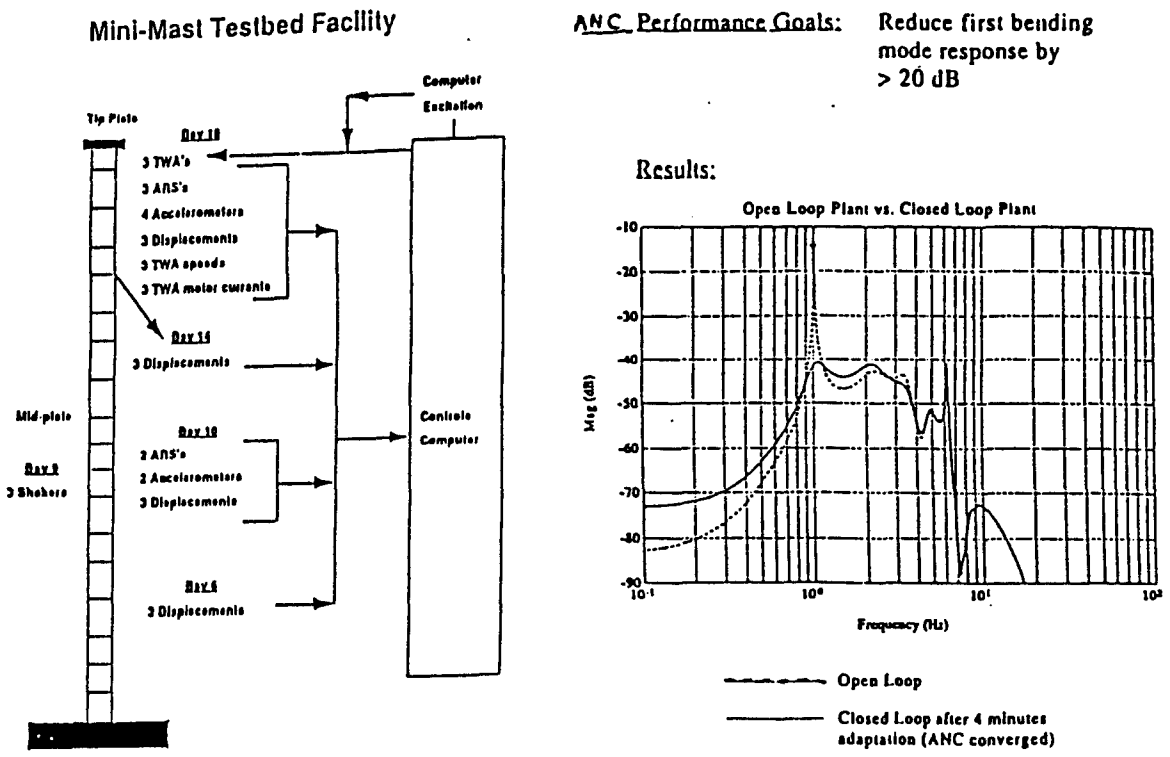


Figure 2-3: Beam Experiment Shows Excellent Convergence Results After 100 Seconds

Multi-Hex Prototype Experiment (MHPE) Addresses Vibration In Large Optics

- 4M Diameter Cassagrain Configuration
- Vibration Control System Uses Nine LPACT Sensor/ Actuator Units

MHPE System ID Results

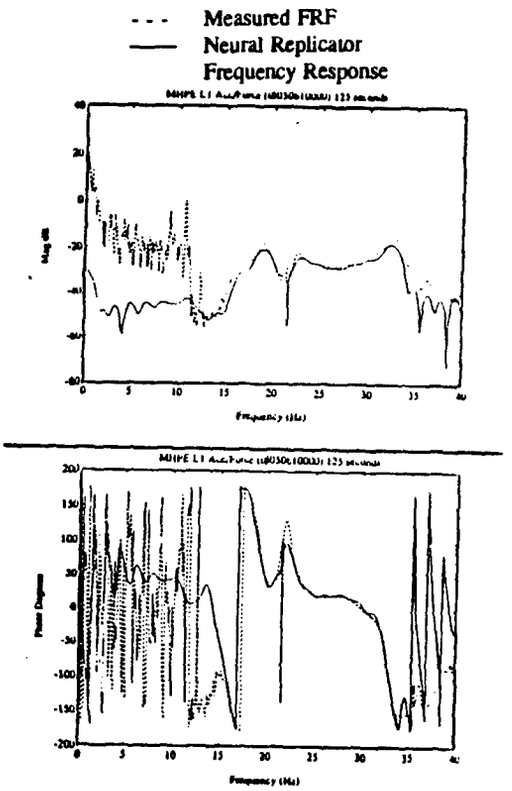
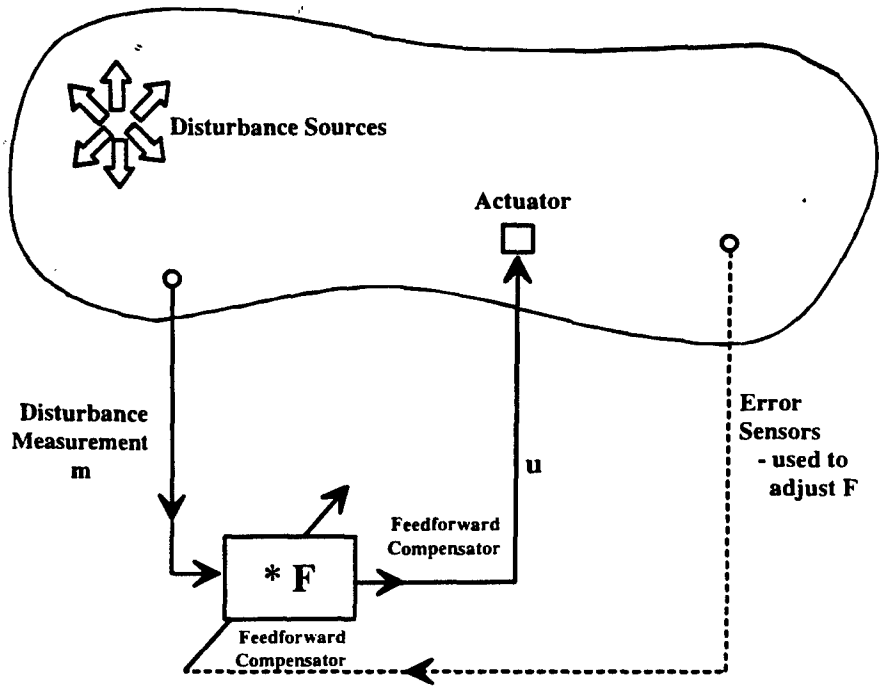
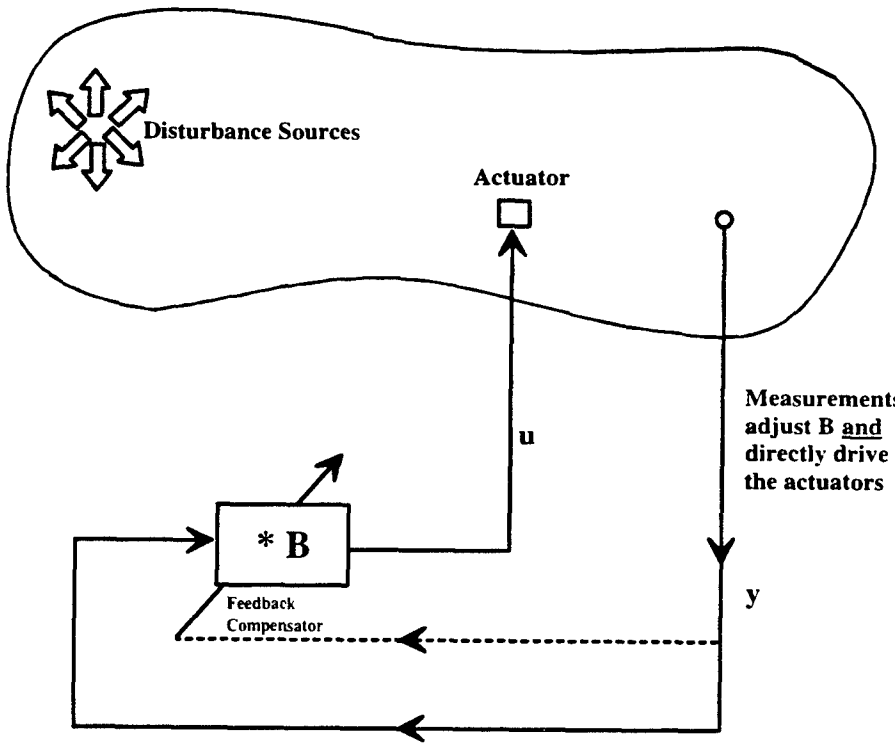


Figure 2-4: Neural Network Closed-Loop System Identification Experiment was Performed on the Dynamically Complex Multi-Hex Prototype Experiment (MHPE) Testbed



Adaptive Feedforward Cancellation

(*denotes convolution)
Once F converges, m directly drives u



Adaptive Feedback Control

- **Comparable or superior performance capabilities**
- **Better transient response**
- **Can simplify control hardware**
- **Can handle disturbances that are inaccessible to measurement**

Once B converges, y drives u

Figure 2-5. Feedforward vs. Feedback Architecture

disturbance measurements (assuming they are available) or a synchronization signal or timing pulse (if the disturbances are periodic). This control architecture is workable only if the fundamental disturbance sources are accessible to measurement. In contrast, adaptive feedback uses the motion measurements not only to adjust the feedback gain B but also to drive the actuators. This is the suitable architecture for systems wherein disturbances are not accessible to direct measurement. Adaptive feedback is a much more challenging problem than feedforward and its successful treatment by ANC is a significant advance. Moreover, adaptive neural controls have been elaborated into a selection of forms capable of handling feedforward or feedback or a combination of the two. This last feature is unique.

In addition, the above time-domain versions of ANC also have frequency-domain counterparts (combining several frequency-domain replicators). In the frequency-domain ANC that combines feedforward and feedback, the two forms of control have synergistic benefits. First feedback control is able to attenuate unmeasurable disturbances and can greatly improve transient response, thereby permitting faster feedforward convergence. Also, closed-loop feedback smoothes out the system transfer functions. This allows the feedforward control to track much more rapid system changes—e.g. disturbance frequency variations due to motor acceleration/deceleration. Alternately, smoother transfer function variations permit a simpler and cheaper feedforward system to be used.

The earliest ANC efforts concentrated on feedback control and many analytical examples were investigated.

Figure 2-6 shows an early example of ANC operation. Using a simulation model of the Mini-MAST facility, an ANC simultaneously performed system identification and control optimization. In this example, the ANC was required to achieve more than an order of magnitude closed-loop attenuation of the first bending mode pair of Mini-MAST leaving higher frequency modes unaltered. This basic control objective was obtained within 7.5 sec. of adaptation and exact agreement with desired closed-loop response was attained after four minutes of adaptation.

In its detailed operation, the ANC carries out a sequence of steps analogous to the modeling ID and design refinement steps carried out by human designers within a streamlined design and test methodology. However, the ANC carries out these steps tremendously faster and without direct human supervision.

The above example helps to illustrate the potentially enormous savings in time and effort for development of initial space structure control design. Note that, by typical performance standards set by the human G.I.s on the Mini-MAST testbed during the CSI GI Program Phase 1, the control design obtained by ANC (Figure 2-6) is quite respectable. However, rather than requiring an elaborate design model together with a man year of effort with several hours of on-site testing, ANC obtains its results without prior information on the testbed and within less than five minutes of unsupervised operation!

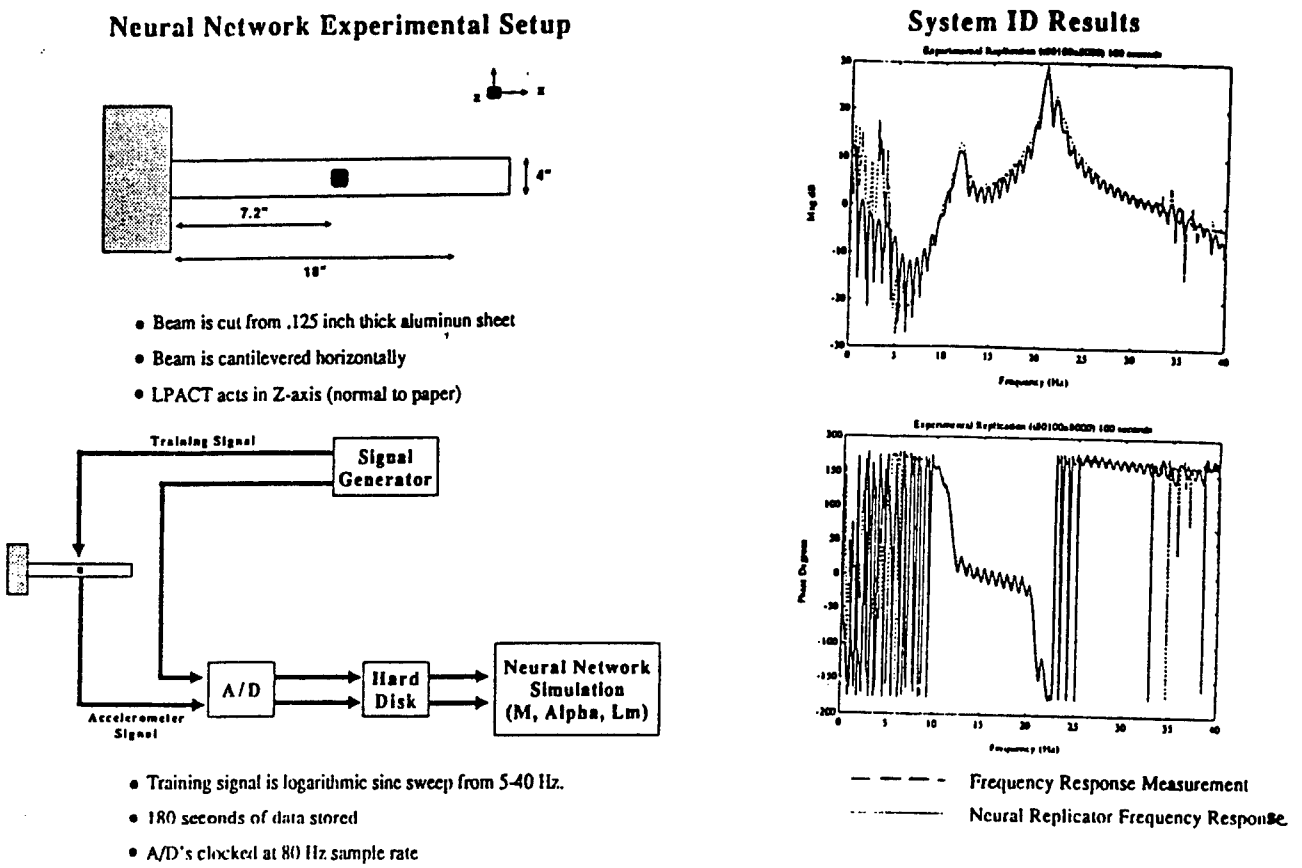


Figure 2-6: Simulation Results for the LaRC Mini-MAST Testbed Demonstrate Simultaneous System Identification and Control Optimization

The above examples stimulated interest in demonstrating the ANC in the laboratory. Recently, feedback control was demonstrated by NASA/LaRC in collaboration with Harris using the Controls Evolutionary Model (CEM) testbed at NASA/LaRC in July 1993. A simplified version of the general (time-domain) ANC model reference adaptive control scheme was used, in which the on-line controller was constrained to be an optimal one-step-ahead controller for the identified plant. The entire algorithm was implemented by programming the lab facility computer. Figure 2-7 summaries the test results. Broadband inputs were used as training stimuli to execute identification and control adaptation. With approximately 10 seconds of training, the system converged to an optimal deadbeat controller.

The above results indicate at least the feasibility of ANC and provided valuable experimental experience with which to avoid potential pitfalls. On the other hand, the experimental tests made

little use of the highly modular, parallel character of the ANC architecture—the algorithm was programmed as a monolithic entity in the lab facility computer, which had capabilities far in excess of that needed. Sizing studies of ANC algorithms have shown that, judging from throughput and I/O bandwidth requirements, the most general ANC can be implemented in a parallel system of several DSP cards, using existing hardware. This configuration is more suitable for practical DoD applications and is a necessity for commercial or dual-use applications.

The development and testing of such DSP-based implementations is the subject of the next chapter.

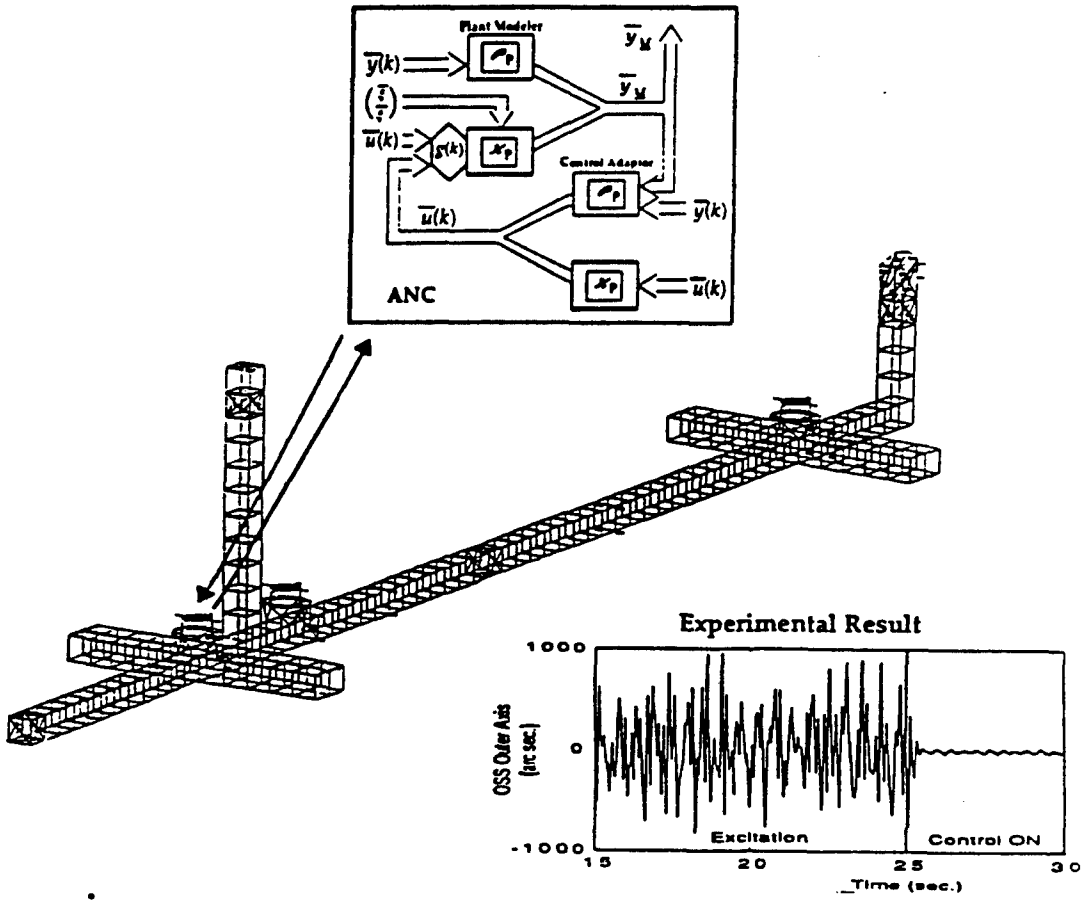


Figure 2-7: In July 1993, NASA/LaRC Personnel Experimentally Demonstrated a Basic ANC System on the NASA/Langley Controls Evolutionary Model

3. Demonstration of Frequency-Domain ANC Algorithms for Feedforward Control

Following the fundamental theoretical and experimental investigations described above (in which the basic feasibility of Adaptive Neural Control was established), development efforts focused on implementation using practical, productizable hardware. It was concluded that a reasonable, near-term hardware implementation suitable for spacecraft and dual use applications involves the use of existing DSP hardware in a multi-processor architecture. Consequently, hardware of this type was used to implement ANC and intensive experimental tests were conducted on a variety of testbeds.

Much of the DSP experimentation was carried out under a Corporate funded project for commercialization of the ANC technology. Thus, this effort concentrated on developing ANC algorithms for DSP hardware that could cost-effectively support commercial noise suppression and vibration control applications. A survey of these applications reveals that the most useful distinction from the point of view of system complexity is the spectral distribution of disturbances. As Figure 3-1 illustrates, there is a hierarchy of disturbance types ranging from harmonic or "single-tone" disturbances, which are the simplest to address, to "broadband" or continuous spectrum disturbances which constitute the most general case and are the most complex.

The harmonic disturbance case refers to those disturbances that are dominated by a single sinusoid, where the amplitude and frequency may vary over time (at rates somewhat below the dominant frequency). It appears that disturbance sources for the majority of commercial applications fall into this class because the objectionable noise or vibration is directly caused by a rotary motor or engine. Examples include fan noise, automobile engine noise, aircraft cabin noise, etc. In many of these cases, the disturbance source—i.e. the engine, motor, rotor or CMG, etc., is directly accessible to measurement, either through a synchronization or timing pulse pick-off directly from the motor or by use of motion sensors located on or very near the source. Consideration of the overall system dynamics shows that the knowledge that the disturbance is harmonic combined with an in-situ timing pulse measurement is tantamount to a direct measurement of the disturbance forces. Consequently, the feedforward control architecture is suitable for suppression of motion due to these disturbances.

At the next level of complexity, multi-tone disturbances consist of two or more harmonics, where again the frequencies and amplitudes of each component may be subject to time variation. This disturbance type is significant in most applications, e.g., nonlinearities in engines and motor mounts produced higher harmonics and sub harmonics in addition to the fundamental tone produced by the rotary source. In many cases, the multi-tone disturbance is truly *periodic*—i.e. all constituent tones are multiples of the fundamental frequency. There are other cases where the constituent tones are *incommensurate*—i.e. although multi-tone as defined above, the disturbance as a whole is periodic. This occurs for many systems of DoD interest. For example, a spacecraft bus many contain several Control Moment Gyros (CMGs) for attitude control Each CMG, due

- **Most applications involve both multi-tone and broadband to some degree**
- **A complete capability must address all 3 types of disturbances**
- **The three disturbance types form a hierarchy of increasing difficulty**

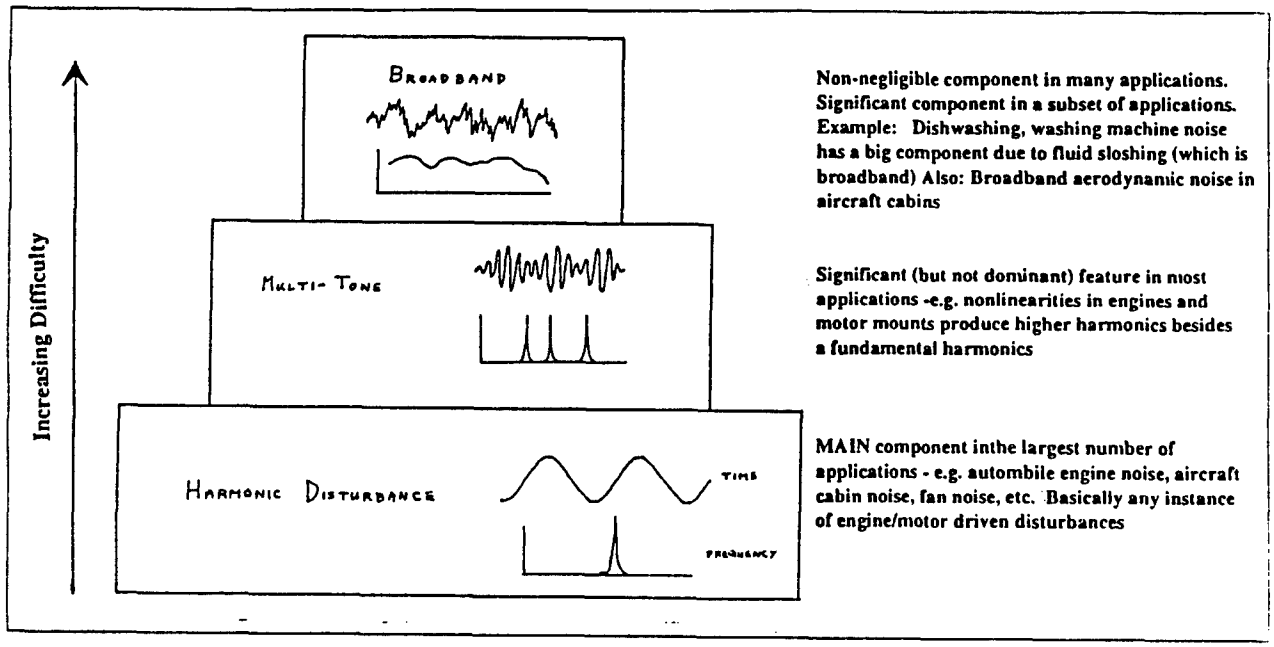


Figure 3-1: Applications of Adaptive Control Involve a Hierarchy of Disturbance Types

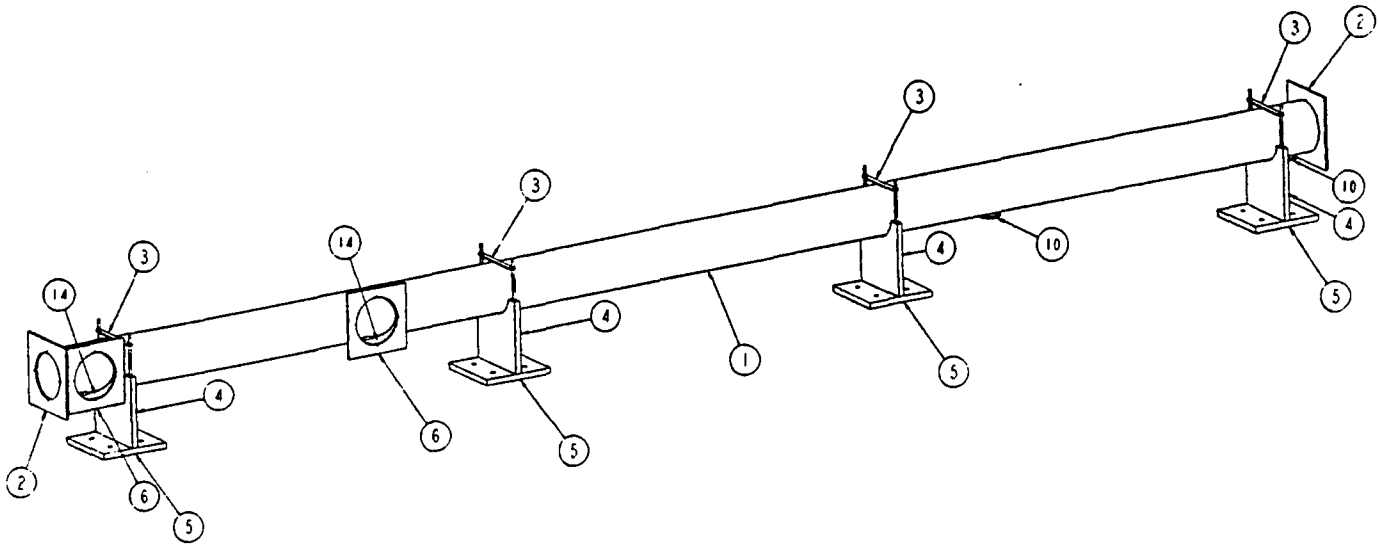
to static and dynamic imbalances produces a primarily harmonic disturbance at its rotor speed. Although nominally identical, each CMG has a slightly different rotor speed. Thus the vibrational disturbances imparted to a sensitive payload mounted on the bus consists of several harmonics with closely spaced frequencies. The periodic case with closely spaced frequencies is particularly difficult to treat and has been the focus of our ANC developments for multi-tone disturbances. Obviously, as in harmonic disturbances the fundamental rotary disturbance sources are also accessible to direct measurement—i.e. one can utilize synch signals taken off each rotary source, etc. In addition, however, in several DoD applications, one may be constrained to use ambient motion measurements that are *not* proximate to the disturbance sources. The successful treatment of such cases has been an objective of the ANC development. Whether the disturbance measurements are obtained directly or indirectly from ambient motion sensors, the feedforward control architecture is the most suitable approach for multi-tone disturbances.

Finally, at the highest level of complexity, broadband (or "continuous spectrum") disturbances are defined as having continuously distributed power spectral densities (PSDs), without discrete tones. This is a nonnegligible component in many applications; e.g. fluid sloshing noise in dish washers and washing machines. This is also a nearly dominant component in a subset of applications; e.g. broadband noise in aircraft cabins produced by aerodynamic turbulence. In many cases, these disturbances are not accessible to direct measurement so that the *feedback* control architecture must be used. In other cases, even though the disturbances are amenable to (at least indirect) measurement, the broad spectral distribution of the disturbances make a *combined* feedforward and feedback control architecture the most efficient, robust approach. The ability to implement adaptive feedback and combined feedforward, feedback control architectures is a unique strength of ANC technology. An example of broadband disturbances that are accessible to measurement is the case of vibration in a large optical structures due to rapid repointing maneuvers—a situation addressed by the Phillips Lab ASTREX testbed. Here the *ultimate* disturbance sources are the *command signals* to the slewing actuators—these are generated by the onboard computer and are obviously directly accessible to the vibration controller for combined feedforward/feedback control.

On the Corporate funded commercialization projects, we implemented a variety of ANC algorithms for harmonic and multi-tone disturbances using a PC interfaced DT3808 DSP card (TI TMS320C40 chip) and developed efficient, rapidly convergent adaptive controls via real-time testing on three entirely different laboratory testbeds. These testbeds are (1) an acoustic duct apparatus, (2) a concentrically mounted proof-mass actuator test rig and (3) the MHPE optical structure vibration experiment apparatus. We first describe the testbeds used as follows:

The Acoustic Duct:

Figure 3-2 shows an engineering drawing of the apparatus and Figure 3-3 shows a simplified block diagram of the testbed. The duct is a 20 ft. long tube with 8 in. inner diameter. It is mounted at one end with a disturbance speaker and the actuator speaker can be placed at two alternate locations. The objective is to suppress noise received at the end of the tube as monitored by an acoustic microphone.



	16			1/2 -20 X 1.000 SOCKET HEAD CAP SCREW
	8			3/8-24 NUT
	AR			3/8-24 ROD
	12			3/8-24 X 1.000 SOCKET HEAD CAP SCREW
	12			4-40 X 0.375 SOCKET HEAD CAP SCREWS
	1			MICROPHONE
	1			SPEAKER
--	1	14	4052-114	TUBE CUTOUT
--	0	13	4052-113	MICROPHONE CAP 2 -- DELETED
--	0	12	4052-112	MICROPHONE INTERFACE 2 -- DELETED
--	3	11	4052-111	MICROPHONE CAP
--	3	10	4052-110	MICROPHONE INTERFACE
--	0	9	4052-109	BENT STRIP -- DELETED
--	0	8	4052-108	SPEAKER BOX ENCLOSURE -- DELETED
--	2	7	4052-107	SPEAKER BOX CAP
A	2	6	4052-106	SPEAKER BOX ENDPLATE
--	4	5	4052-105	STAND PLATE 2
--	4	4	4052-104	STAND PLATE 1
--	4	3	4052-103	TUBE STRAP
A	2	2	4052-102	TUBE ENDCAP
A	1	1	4052-101	TUBE

Figure 3-2: The Acoustic Duct Apparatus was Used to Investigate Adaptive Control of One Dimensional Acoustic Systems

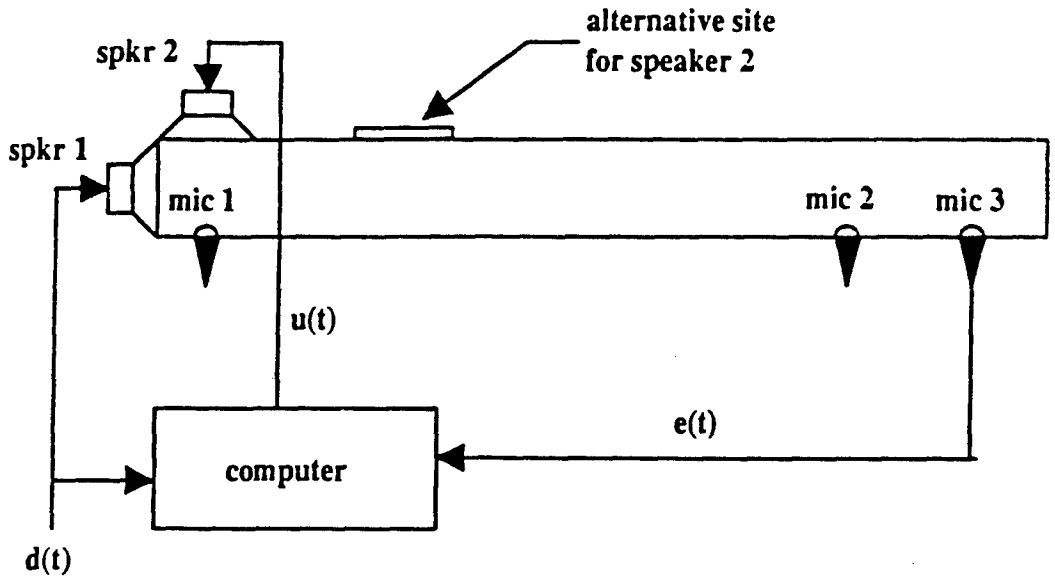


Figure 3-3: Schematic Diagram of the Adaptive Control System for the Acoustic Duct Apparatus

Concentrically Mounted Actuator Test Rig

The concentrically mounted proof-mass actuator test rig is shown in Figure 3-4. A base plate is connected by a tubular structural member to a star-shaped test body (at the top of Figure 3-4) having nontrivial dynamic characteristics. The system is disturbed by up to three shakers (labeled dist #1 etc. in Figure 3-4) mounted on the base plate. A proof-mass actuator, having a toroidal configuration is clamped around the lateral surface of the tube in order to provide control forces. The objective is to suppress vibration in the star shaped body at the top of the test rig, as monitored by an accelerometer at the top of the tube.

MHPE Apparatus

The optical structure testbed is the Harris Multi-Hex Prototype Experiment (MHPE) apparatus, constructed under a Harris IR&D program in 1988. Figure 3-5 shows a simplified block diagram of the MHPE and the test setup. The MHPE has the form of a Cassagrain telescope and consists of a base plate (supported by air bags) connected by a six member truss to the primary mirror support structure, holding a tripod tower which supports the secondary mirror assembly. Disturbances are injected using a proof-mass actuator affixed to the primary mirror support structure. Three additional proof-mass actuators with colocated accelerometers are mounted on the secondary mirror platform to provide actuation and motion sensing. The objective is to suppress vibrational accelerations at the secondary mirror platform that are excited by the forces applied to the primary support structure.

For feedforward cancellation of single-tone disturbances in all of the above three testbeds, we specialized the feedforward, frequency-domain version of the ANC algorithm. In contrast with basic LMS algorithms, the ANC system for this case is able to simultaneously identify all needed transfer functions and simultaneously adapt the actuator inputs without interrupting normal operation or injecting an extraneous, broadband test signal (dither). Moreover, the ANC algorithm is fast: with no previous identification, completed cancellation is achieved in 3 to 5 iterations (10-15 seconds in these experiments); once transfer coefficients are identified, control adjustment can be accomplished in one step.

The single-tone ANC algorithm was experimentally demonstrated on all three testbeds. These live demonstrations are documented in the Harris Corporation video "Adaptive Noise and Vibration Cancellation Demo". Starting from a clean slate (no prior identification data) the system converges in a fraction of a second. The video shows how ANC can track smooth changes in system dynamics, and can quickly recover after dramatic, sudden system changes. Moreover, the same algorithm, without modification is shown to work on many different types of systems. Over 20 dB noise or vibration reduction is achieved in all cases.

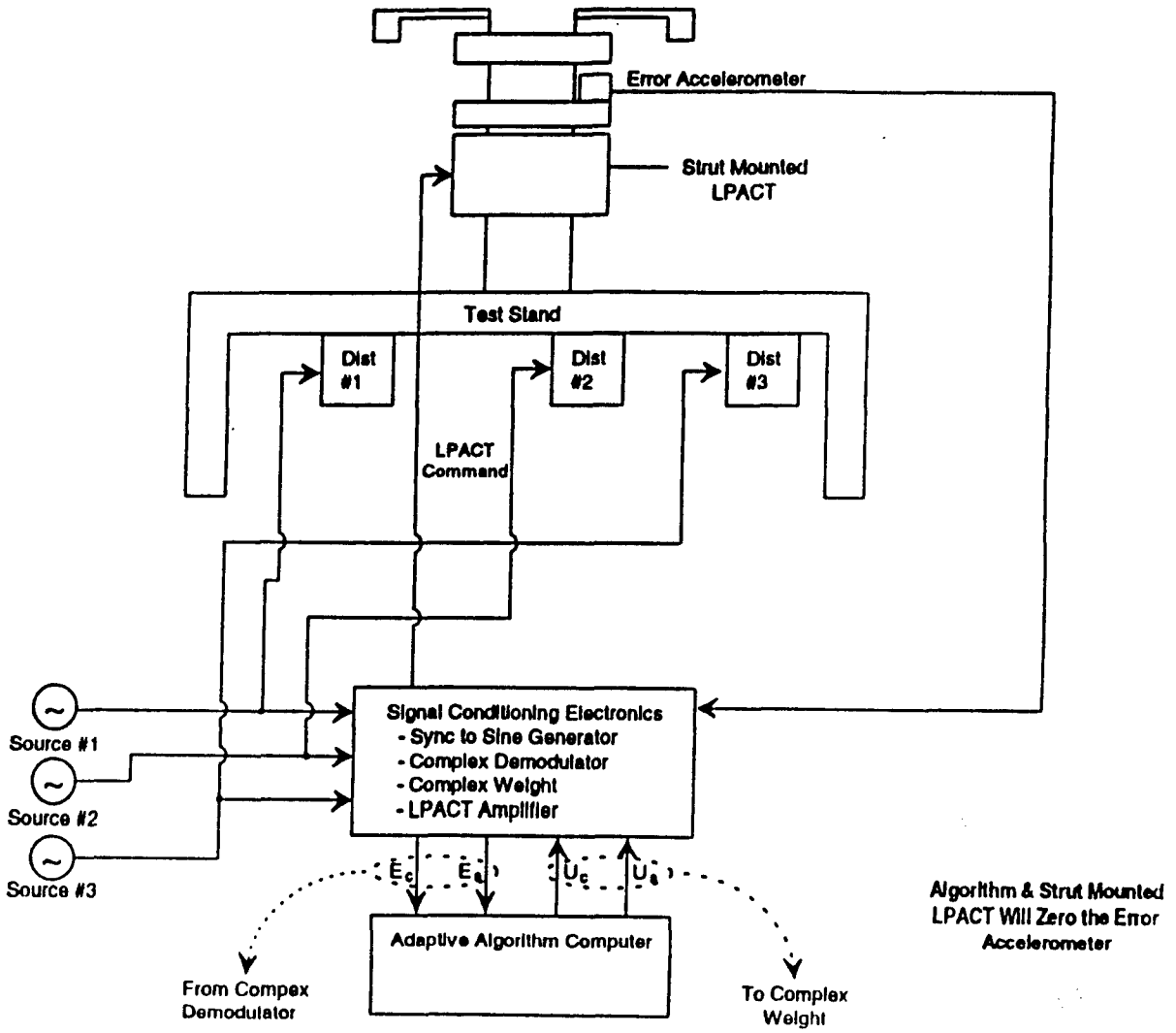


Figure 3-4: Force Cancellation System Demonstration

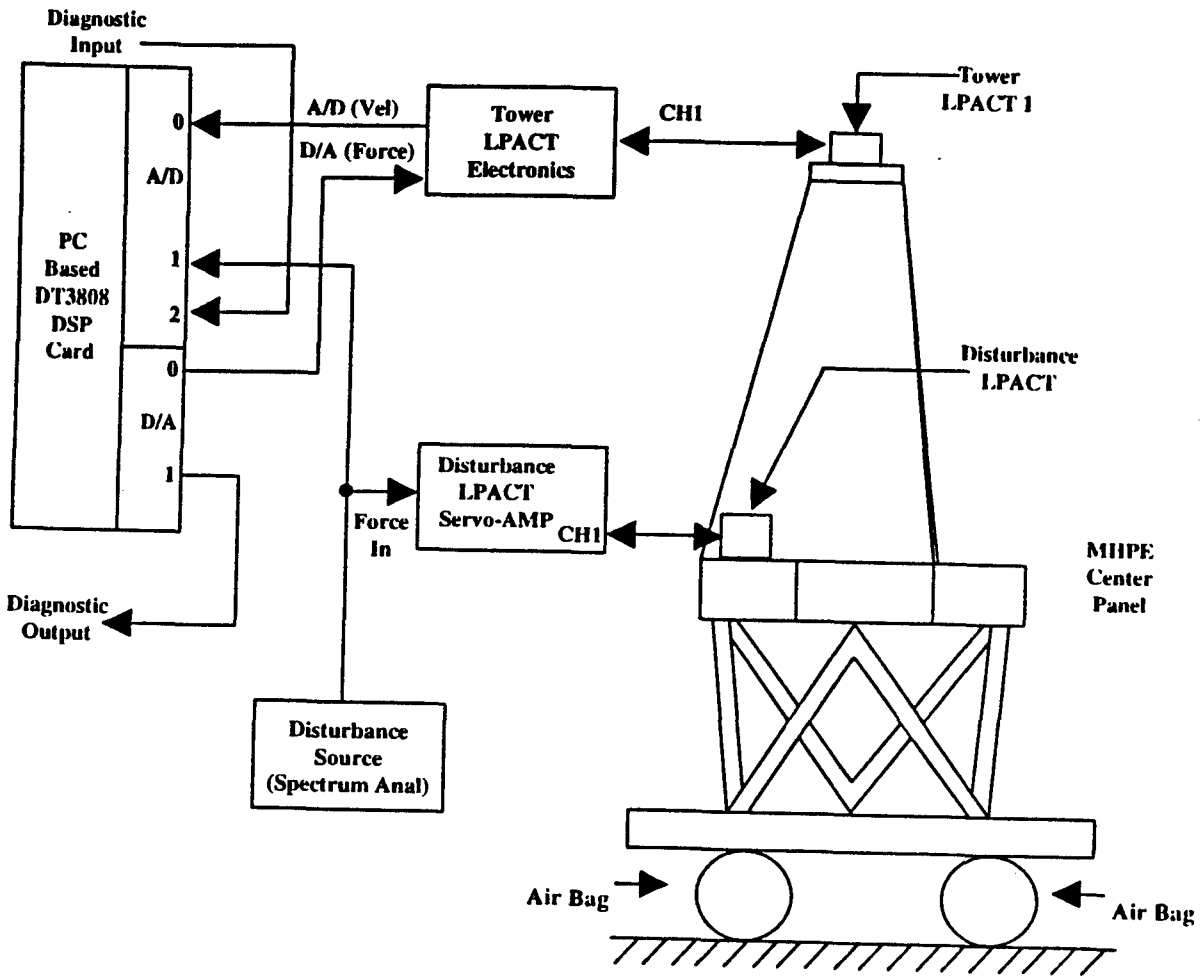


Figure 3-5: System Block Diagram for Multi Hex Prototype Experiment (MHPE) Demonstration of Harmonic Vibration Cancellation

Subsequently, the ANC single-tone algorithm was extended to handle multiple tones. The ANC system for this case combines a fast neural demodulator unit with a array of single tone canellors. Multi-tone cancellation is routinely demonstrated in the laboratory. For example, Figure 3-6 shows results pertaining to the concentrically mounted actuator test rig for two relatively closely spaced harmonics. Over 30 dB and approximately 40 dB attenuation is achieved on the two tones, respectively. Starting with no prior transfer coefficient information, simultaneous identification and control are achieved in approximately one second. For the same testbed, Figure 3-7 illustrates results for three closely-spaced tones. The algorithm again converges in approximately one second, achieving 20 dB attenuation for all tones.

The above demonstrations revealed some unique features in our ANC, frequency-domain algorithms. As noted, one unique feature is the simultaneous control adaptation and system identification without test signals and without the need for look-up tables. Although this was not done on the Corporate-supported projects, the potential of ANC to combine feedforward with feedback control is also unique.

With the above background, we proceeded on the ANC program to investigate feedforward control for continuous spectrum disturbances—considering both time-domain and frequency-domain versions of ANC.

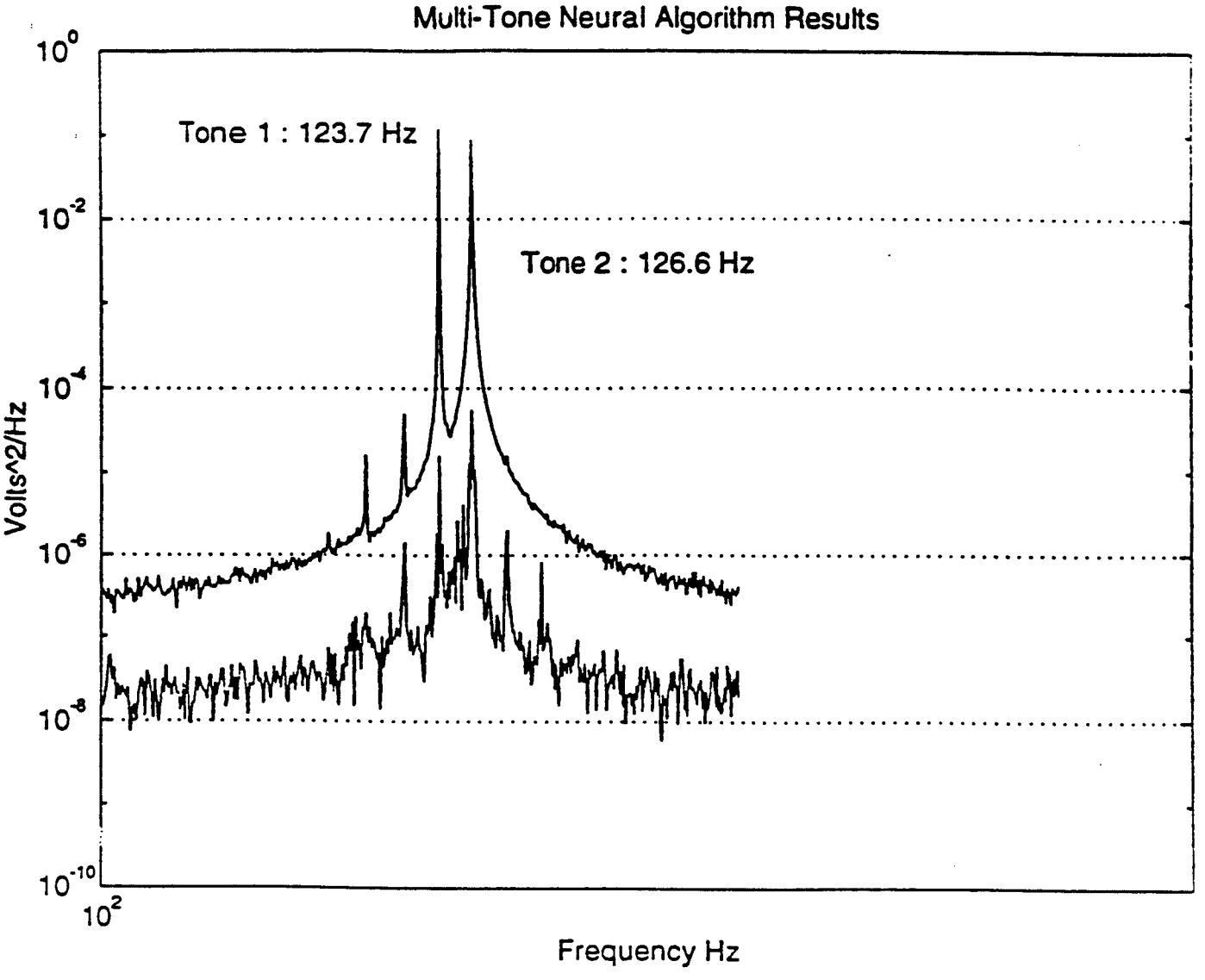


Figure 3-6: Multi-Tone Neural Algorithm Results for the CMLPACT Test Rig

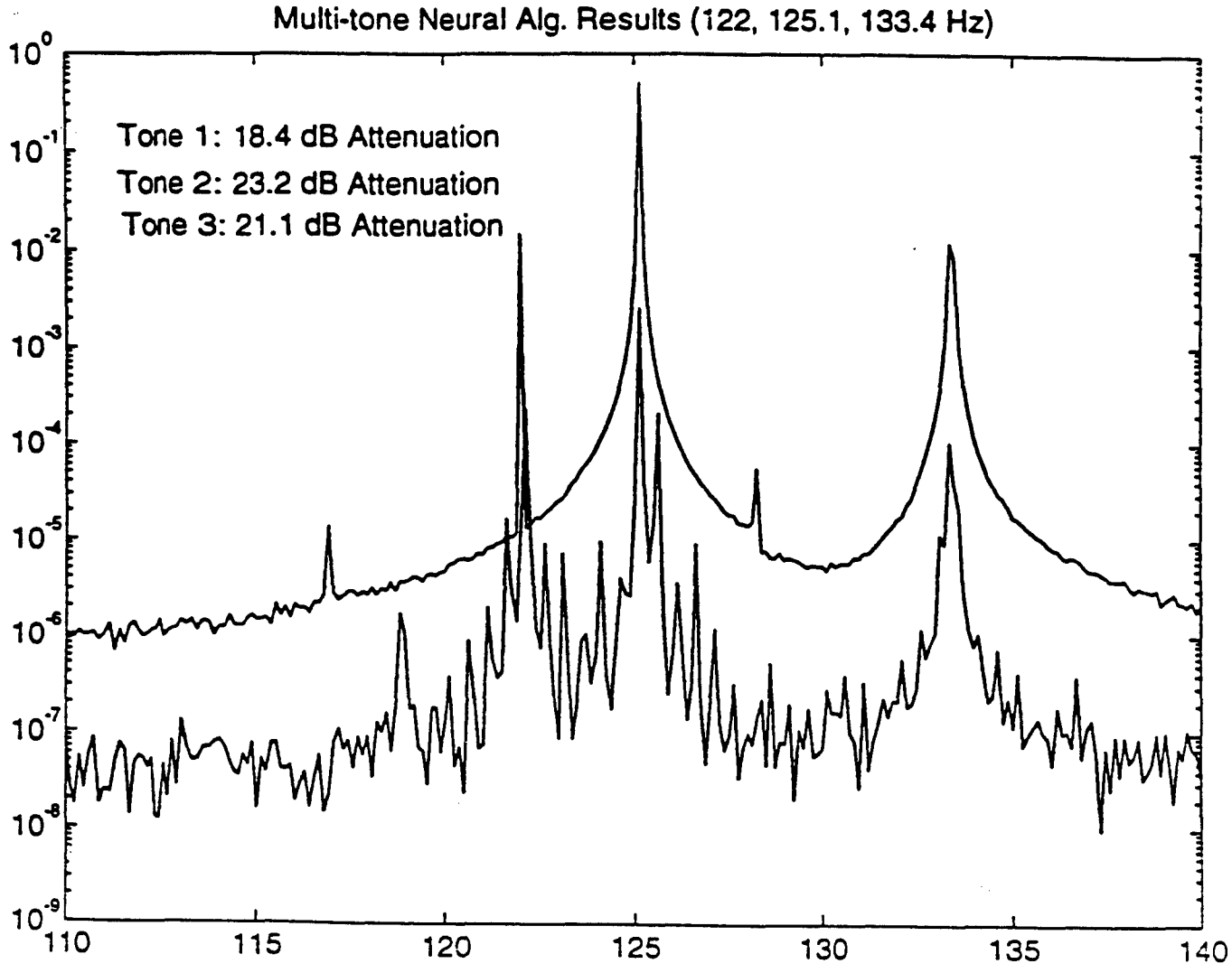


Figure 3-7: Multi-Tone Neural Algorithms Results (122, 125.1, 133.4 Hz) for the CMLPACT Test Rig

4. ANC Program Developments

4.1 Introduction

The Basic segment of the ANC program is essentially a preparation for experimentation on the ASTREX testbed (or a suitable alternative) with the objective being to suppress vibrations excited by repointing maneuvers. These disturbances are typically continuous spectrum, not tonal. As noted for the repointing situation in the previous chapter the ultimate disturbance sources are the command signals to the slewing actuators and these are directly accessible to the vibration controller. Accordingly a feedforward control architecture is appropriate. Hence on the ANC Basic program we concentrated on feedforward control of continuous spectrum disturbances. We studied various issues associated with the use of DSP hardware using the MHPE testbed. The objective of this phase is to successfully demonstrate adaptive control of the MHPE testbed with an ANC algorithm that can be readily scaled up to handle larger scale test facilities.

So that we may give a more detailed description, Figure 4.1-1 again shows the test facility used to-date. The MHPE without its outboard panels is mounted on air bags in the inflated condition. A Linear Precision Actuator (LPACT), proof-mass actuator is mounted on the center panel to provide broadband disturbances emulating the vibrational disturbances that might be excited by equipment or rapid slewing. The control objective is to suppress lateral motion of the secondary mirror platform since this is directly proportional to line-of-sight (LOS) errors. Accordingly, only the LPACTs on the secondary mirror platform are used for control and tower bending motion is sensed by the colocated accelerometers. These actuators and sensors interface with a PC based DT 3808 DSP card, as does a spectrum analyzer-generated disturbance source.

Using this test set-up, two main issues were explored. Although, in the tonal cancellation work described in the proceeding chapter, a frequency-domain version of ANC was the obvious choice, both time-domain and frequency-domain forms can be used for continuous spectrum disturbances. Thus our first objective was to evaluate both time and frequency-domain versions of ANC algorithms. Secondly, while throughput, I/O bandwidth and memory capacity requirements have been carefully studied for general ANC algorithms, the design limitations imposed by instrumentation noise and machine accuracy; i.e. word length, needed further study. Thus, a second objective was to shed more light on these issues by means of extensive experimentation with the DSP hardware.

Our approach in conducting these studies was to closely enmesh theoretical deductions with numerous and frequent experimental trials. Table 4.1 summarizes in chronological order the thirteen major experimental tests that were performed using both time and frequency-domain versions of ANC, before the recommended configuration was arrived at. We devote much of the remainder of this chapter to the description of these tests and their results. For greater clarity, we depart from chronological order and describe the tests involving the time-domain and frequency-domain versions of ANC modules separately in the following two subsections.

4.2 Testing of Time-Domain Algorithms

Referring to Table 4.1, test ANC 1 conducted initial tests of the basic segments of code comprising the time-domain, ARMA or ARMarkov-based neural replicators. Following success of these code segments on simple test cases, a SISO ARMA replicator was tried in ANC 2. For further reference, Figure 4.2-1 gives simplified block diagram conventions for description of the time-domain neural replicators. The simple block at the lower left of the figure represents an FIR time-domain replicator (given in expanded ANC notation at the lower right). Note that the adaptive speed in this block is replaced by unity so that in the diagrams that follow we can indicate the adaptive speed explicitly by multiplying the error signal by the appropriate factor. ANC 2 implements an ARMA-based replicator consisting of two such FIR blocks, with inputs equal to the disturbance system input and error sensor (one of the accelerometers) output and outputs summed to form the replicator output. Using a shaker at the MHPE base, good agreement was obtained with FRF data. With this success, ANC 3 extended the ANC 2 scheme to a 2 input/2 output MIMO replicator. After chasing down an unaccounted for time delay in the system, ANC succeeded in converging. However approximately 3 to 5 minutes was required for convergence. The reasons for the sluggish behavior were finally determined only after a diagnosis (as discussed below) of ANC 10 results.

In any case, after ANC 3 it was judged time to add feedforward control to the identifier function in ANC 4. ANC 4 is shown in Figure 4.2-2. The upper portion of the figure shows an ARMA-type replicator where the replicator forward-path output has the form:

$$\hat{y}(k) = \sum_{m=1}^{n} \left[F_m y(k-m) + G_m d(k-m) + P_m u(k-m) \right]$$

$$\begin{aligned} y &= \text{error sensor output} \\ d &= \text{disturbance input} \\ u &= \text{control actuator command} \end{aligned} \qquad (4.2\text{-}1)$$

The "algebraically constrained" ARMA controller is a deadbeat controller formed from the identified ARMA coefficient vectors F, G and P. Note, from the bottom half of Figure 4.2-2 that the control command, u, at time k is given by:

$$u(k) = -\frac{1}{P_1}\left[\sum_{m=1}^{n} \left[F_m y(k+1-m) + G_m d(k+1-m) \right] + \sum_{m=2}^{n} P_m u(k+1-m) \right] \qquad (4.2\text{-}2)$$

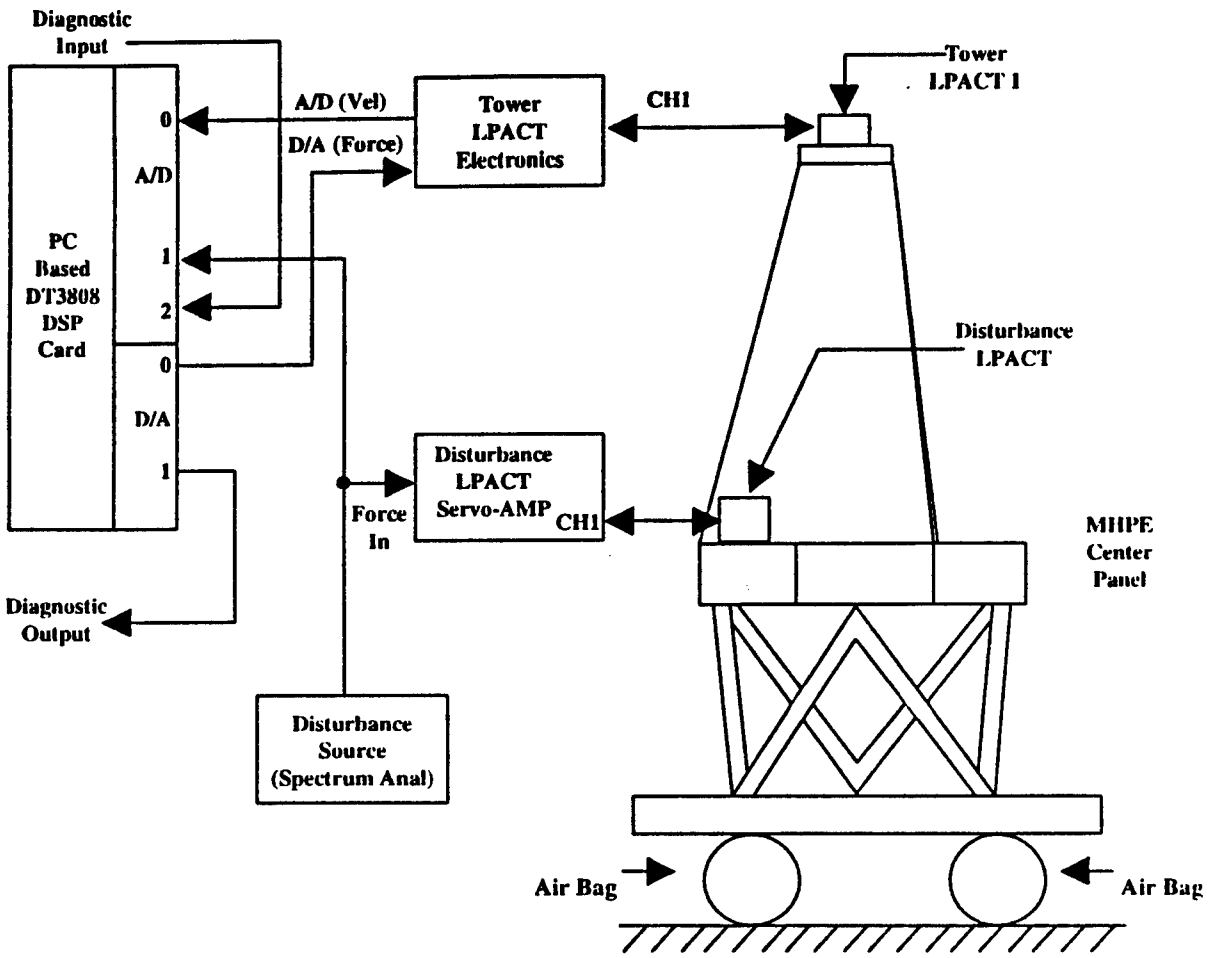


Figure 4.1-1: System Block Diagram of ANC Experimental Set Up

Table 4.1: Candidate ANC Designs—Test Experience

Program	Configuration	Result
ANC 1	Off-line checkout of delay line replicator code segments	
ANC 2	SISO series-parallel ID	Used a shaker input, got good match of I/O behavior using feedback system output.
ANC 3	2 x 2 MIMO series-parallel ID	Required 3-5 minutes to converge.
ANC 4	2 x 1 MISO ID with algebraically constrained ARMA control	Unstable - puzzling because of apparent good input/output behavior match prior to control. Note possibility of non-minimum phase model. Hard to debug (but see ANC 10 below).
ANC 5	Off-line checkout of FFT/replicator code segments	
ANC 6	SISO FFT/replicator ID	Matched primary path well with a 1 Hz resolution.
ANC 7	2 x 1 MISO ID with FFT/replicator, control via division of identified primary and secondary path weights to form FFT based controller	Distributed 5-30 Hz band, error and drift in the weights below 5 Hz caused large drifts in control => no appreciable attenuation overall.
ANC 8	2 x 1 MISO ID, control via pre-filtered replicator adaptation of FFT controller weights	Drifting/unstable controller adaptation.
ANC 9	SISO FFT weight filter with fixed, user-selected FRF	Verified ability to implement a specified FRF with this type of FFT filter.
ANC 10	2 x 1 MISO series-parallel ID, control via Explicit FRF computation and assignment to FFT weight controller	Revealed serious roundoff concerns with weight updates and delay-line filter computations.
ANC 11	SISO ID of a single frequency bucket using cross and auto spectral density averaging based on recursive FFT	Computes good results as compared with spectrum analyzer, but would like a little more speed.
ANC 12	SISO ID of multiple frequency buckets using ANC 11 algorithm, estimates accuracy of each bucket ID, adjusts FFT control of each bucket independently	Good results, desire increased speed and decreased complexity. Also, improved rigor needed in ID (really need twice the resolution of control FFT in order to improve reliability).
ANC 13	Simple SISO FFT replicator with adjustable learning rate for ID (twice resolution of control), control algorithm requires no ID accuracy estimate	Reliable convergence on center of channel, but poor performance in between.
ANC 14	Replicator with adjustable learning rate also used for control updates	Reliable convergence, good overall performance compromise achieved over whole frequency band.

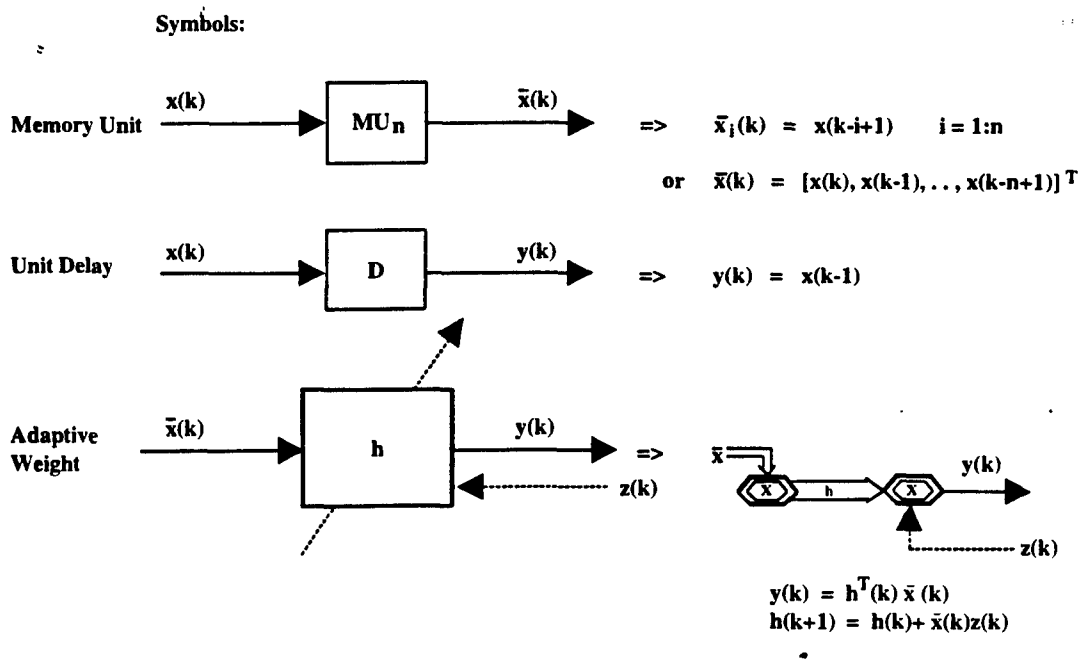


Figure 4.2-1: Block Diagram Conventions for Time-Domain, ARMA-Based Neural Replicators

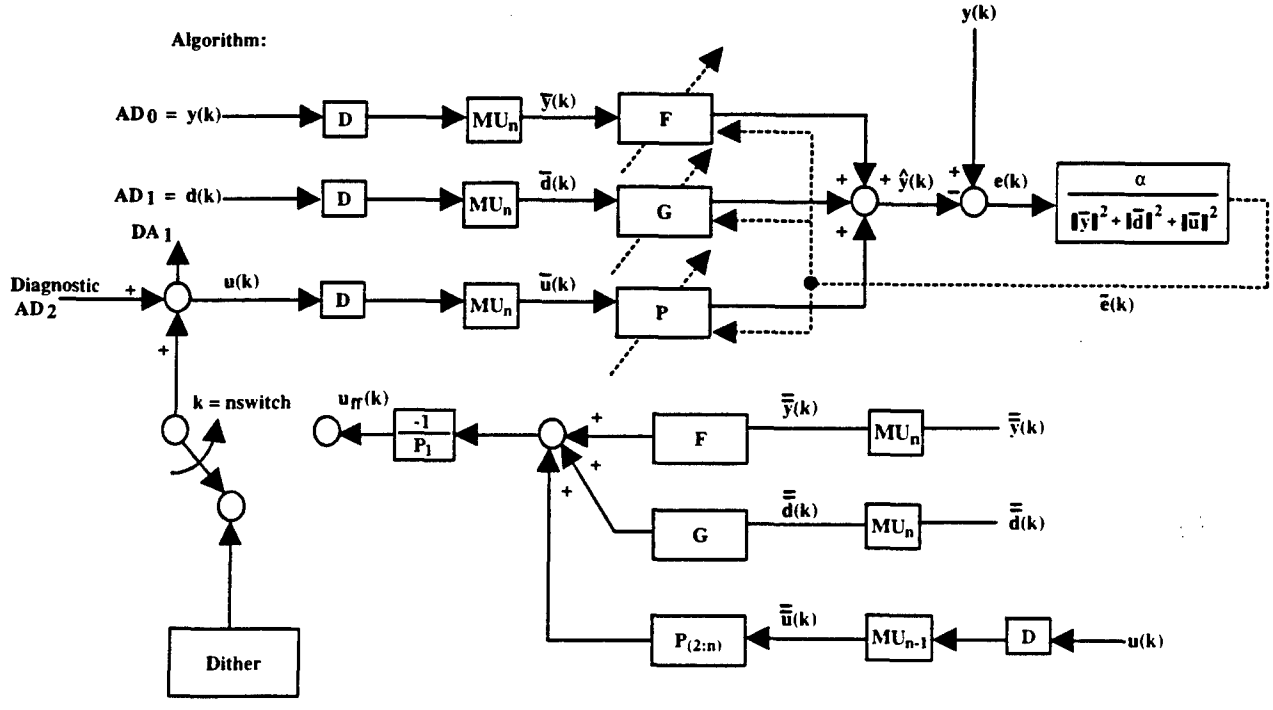


Figure 4.2-2: ANC 4, Series-Parallel ID Using ARMA Model and Algebraically Constrained ARMA Control

When this is substituted into (4.2-1), it is seen that $\hat{y}(k) = 0$. Thus, once F, G and P converge to the actual ARMA coefficients of the system, one expects that $y(k)$ converges to zero. As indicated by the switch shown at the lower left of Figure 4.2-2, a random signal is first injected to

the actuator command in order to perform system identification, then u_{ff} is switched on to execute control. Although the identification step was apparently successful and good matching of input/output behavior was obtained, $y(k)$ did not converge to zero. This puzzling behavior was thought to be the result of identification errors leading to a non-minimum phase model. However, as in ANC 3, the fundamental cause of the aberrant behavior was not properly diagnosed until the results of ANC 10 were understood.

We attempted in ANC 10 one last try to implement a time-domain type replicator and to diagnose the results obtained in ANC 4. In the following we consider ANC 10 in detail because the results did lead to an understanding of the time-domain results and reveal a significant design issue with ARMA-based replicators.

The algorithm tested in ANC 10 is shown in Figure 4.2-3. The replicator is a standard ARMarkov replicator with delay period L. The controller is synthesized from the ratio of the FFT (see the following section, equation (4.3-1)) of the forward path impulse response, G, to that of the secondary path, P. The FFT based filter for generating u had been extensively verified on earlier tests (described in the next section) so that any problems encountered could be attributed to the replicator. By focusing

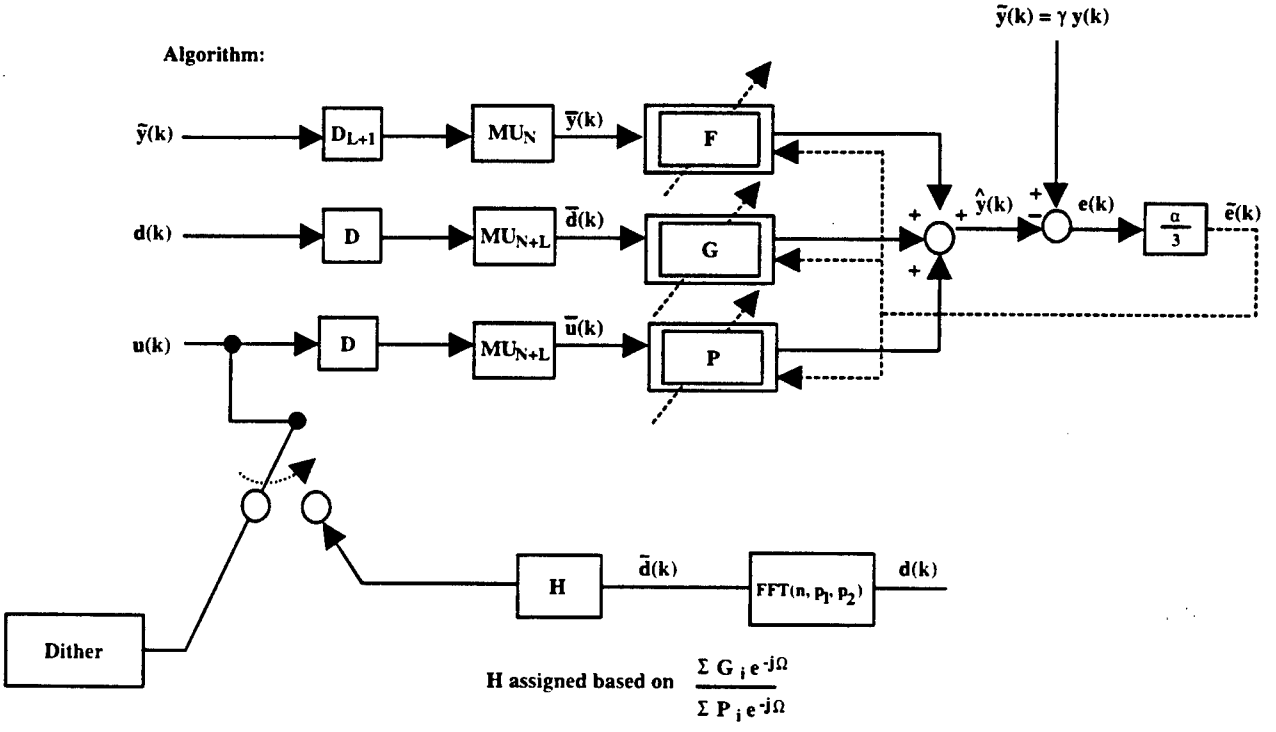


Figure 4.2-3: ANC 10 Series/Parallel Plant I.D. Control via Explicit Computation of Cancellation FRF

on the replicator operation and acquiring a wide variety of diagnostic outputs during the test, we were able to determine that both the weight updates and forward path signal computations suffered from serious roundoff errors that prevent controller convergence. Use of larger delay periods, L, in the ARMarkov model formulation quantitatively reduced the degree of error but did not qualitatively eliminate the phenomenon. The sensitivity of time-domain replicators (both

ARMA and ARMarkov) to roundoff error due to finite machine precision was then verified for the conditions of ANC 3 and ANC 4 and the same qualitative phenomena were duplicated on simple test circuits and correlated with analysis.

The reasons for ARMA or ARMarkov replicator sensitivity to machine precision can be explained as follows. The problem shows up even in the forward path calculations because of inherent ill-conditioning of ARMA-type filters for high sample rates and systems composed of very lightly damped modes with frequencies well below the sample rate. To explain this, first note that a multi-mode structural model with forcing input *x(k)* and sample data sensor output *y(k)* has the ARMA-like representation:

$$
\begin{aligned}
y(k) &= \sum_{m=1}^{N} y_m(k) \\
y_m(k+1) &= \left[2\,\rho_m c.\theta_m, -\rho_m^2\right] \begin{pmatrix} y_m(k) \\ y_m(k-1) \end{pmatrix} \\
&+ \left[\gamma_m, \rho_m\left(\alpha_m s.\theta_m - \gamma_m\, c.\theta_m\right)\right] \begin{pmatrix} x(k) \\ x(k-1) \end{pmatrix}
\end{aligned}
\tag{4.2-3}
$$

where α_m, γ_m are modal influence coefficients *c.( ) and s.( ) denote cos(.) and sin(.)* y_m represents the contribution of mode *m* to *y* and:

$$
\begin{aligned}
\theta_m &= 2\pi \frac{\Omega_m}{\Omega_s} \\
\Omega_m &= m^{th} \text{ mode frequency} \\
\Omega_s &= \text{sample frequency} \\
\rho_m &= \exp\left[-\eta_m \frac{\Omega_m}{\Omega_s} 2\pi\right] \\
\eta_m &= m^{th} \text{ mode damping ratio}
\end{aligned}
\tag{4.2-4}
$$

The above expressions can be manipulated to produce an overall ARMA model. To give an explicit example, consider two modes with $\alpha_1 = \alpha_2 = \alpha, \gamma_1 = \gamma_2 = 0$ and $\rho_1 = \rho_2 = \rho$. We get the ARMA model:

$$
\begin{aligned}
y(k+1) &= \left[4\rho^2\left(c.\theta_1 + c.\theta_2\right)^2 - 2\rho^2\left(1 + 2c.\theta_1 c.\theta_2\right)\right] y(k-1) \\
&+ \left[-4\rho^3\left(c.\theta_1 + c.\theta_2\right)\left(1 + 2c.\theta_1 c.\theta_2\right) + 2\rho^3\left(c.\theta_1 + c.\theta_2\right)\right] y(k-2) \\
&+ \left[4\rho^4\left(c.\theta_1 + c.\theta_2\right)^2 - \rho^4\right] y(k-3) - 2\rho^5\left(c.\theta_1 + c.\theta_2\right) y(k-4) \\
&+ \rho\left(s.\theta_1 + s.\theta_2\right) x(k-1) \\
&+ \left[2\rho^2\left(c.\theta_1 + c.\theta_2\right)\left(s.\theta_1 + s.\theta_2\right) - 2\rho^2 s.\left(\theta_1 + \theta_2\right)\right] x(k-2) \\
&+ \left[-4\rho^3\left(c.\theta_1 + c.\theta_2\right) s.\left(\theta_1 + \theta_2\right) + \rho^3\left(s.\theta_1 + s.\theta_2\right)\right] x(k-3) \\
&+ 2\rho^4\left(c.\theta_1 + c.\theta_2\right)\left(s.\theta_1 + s.\theta_2\right) x(k-4)
\end{aligned}
\tag{4.2-5}
$$

Now consider what is actually computed at each time step when implementing the above as a digital filter. In particular we concentrate on the disturbance input terms (in x). When the disturbance frequency band is well below the sample rate and Ω_s is large, *x(k-1-m)* can be approximated by *x(k-1)* so that input terms are basically the sum of the ARMA coefficients of x. All of the significant coefficients are of the same order of magnitude (same exponent) but their sum is very small. Indeed, using small angle approximations for the sines and cosines, the net contribution of the disturbance terms to *y(k+1)* is approximately:

$$\begin{aligned} & 5\left(\theta_1+\theta_2\right)\left(1-\rho\right)^2\ x(k-1) \\ & \approx\ 10\pi\left(\frac{\Omega_1+\Omega_2}{\Omega_s}\right)\left(\eta_1\frac{\Omega_1}{\Omega_s}2\pi\right)^2\ x(k-1) \end{aligned} \tag{4.2-6}$$

Thus, if Ω_s is very large compared to Ω_1 and Ω_2 and η is small (of the order 1% or less) then the input terms in the ARMA model will sum to a very small number less than the last significant digit retained and the sum will be rounded off to zero. This predicts that the ARMA filter may work at a sufficiently low sample rate but produce zero output for nonzero input for larger sample rates. The same kind of phenomenon should occur as modal damping ratios are reduced. These predictions were entirely verified experimentally by simulating ARMA filters programmed into the DSP card by the outputs of analog filters set up to emulate three modes at 10, 20 and 30 Hz with 1% damping. The ARMA filter implemented at 1000 Hz produced no discernible output. The modal damping ratios were also varied to produce the behavior predicted.

In the general case, for ARMA models of the form:

$$y(k)=\sum_{i=1}^{n} a_i x(k-i)-\sum_{i=1}^{n} b_i y(k-i) \tag{4.2-7}$$

the transfer function:

$$F(z)=\left(a_0 z^n+\ldots+a_n\right)\ /\ \left(z^n+b_1 z^{n-1}+\ldots+b_n\right) \tag{4.2-8}$$

has its zeros and poles clustered near *z=1* for very light damping and high sample rate. As in the two mode case, the distance of the zeros from *z=1* is proportional to the damping and inversely proportional to sample rate. This can generally be expected to produce significant roundoff errors in accumulating the input terms in the case of multi-mode lightly damped systems. In addition to the forward path signal computations, similar roundoff error problems were identified in the weight update computations.

In summary, the limitations imposed by machine precision on successful implementation of time-domain, ARMA-based ANC algorithms was finally revealed by our combined theoretical and experimental investigation. In the light of these results we considered three options for further development. One option is to retain the time-domain ANC formulation but reduce the digital system sample rate used on the MHPE experiments. This could potentially circumvent the machine precision issue but would also significantly reduce the controller bandwidth. A second option is to again retain the time-domain formulation but somehow increase the processor word length. However, this would entail the procurement of much more expensive hardware that is

inconsistent with later transitions of the technology into commercial applications. The third option and the one selected is to implement and test the frequency-domain ANC algorithms. As discussed in the next section, tests of the basic frequency-domain modules, particularly in ANC 9 showed no appreciable sensitivity to machine precision. Furthermore the frequency-domain ANC formulation meshes well with both classical and modern frequency-domain design insights.

4.3 Test Results for Frequency-Domain ANC Algorithms

Referring to Table 4.1, we initiated in ANC 5 experimental investigations of frequency-domain versions of the basic ANC modules. This sequence encompassed ANC 5–9 and ANC 11–13.

Frequency-domain versions of the ANC components are formed from time-domain replicators by insertion of the finite-Fourier transform (FFT) operator at the inputs or the outputs. If $x(k)$ is a discrete time sequence, define the FFT as:

$$\mathcal{F}_r[x(k)] \triangleq \frac{1}{M}\sum_{m=1}^{M} e^{-ir(m-1)\frac{2\pi}{M}}\, x(k-m+1) \tag{4.3-1}$$

$$M = 2^N \text{ for some integer N}$$
$$r = 0, 1, \ldots, M-1$$

The inverse relation is:

$$x(k-m+1) = \sum_{r-0}^{M-1} e^{-ir(m-1)\frac{2\pi}{M}}\, \mathcal{F}_r[x(k)] \tag{4.3-2.a}$$

and, in particular:

$$x(k) = \sum_{r-0}^{M-1} \mathcal{F}_r[x(k)] \tag{4.3-2.b}$$

The vector $\left(\mathcal{F}_0[x(k)], \mathcal{F}_1[x(k], \ldots, \mathcal{F}_{M-1}[x(k]\right)^T$ is obviously related to $\left(x(k), x(k-1), \ldots, x(k-M+1)\right)^T$ by a unitary matrix operator. This operator can be inserted into the inputs to a time-domain replicator and the conditions for convergence remain essentially unaltered. However, the nature of the approximation changes. For example, if a system with output y and input x is FIR, then a time-domain replicator would be based upon the (forward path) dynamic model:

$$y(k) = \sum_{m=0}^{M-1} h_m\, x(k-m); \quad M = 2^N \tag{4.3-3.a}$$

where the h_m are the Markov parameters. But we can also use(4.3-2.a) to write this in the form:

$$y(k) = \sum_{r=0}^{M-1} H_r\, \mathcal{F}_r[x(k)] \tag{4.3-3.b}$$

This translates into a replicator in which x is first submitted to an FFT operation, then input to a ganglion connected to the output neuron with weight vector $[H_0, \ldots, H_{M-1}]$. Assuming the ganglia and synaptic connections are constructed according to established ANC rules, convergence is readily assured and (4.3-3.a) and (4.3-3.b) would seem to be entirely equivalent. However, in the frequency-domain version, the weights H_r are now the values of the system frequency response function at the FFT (discrete time) frequency points $\theta_r = r\frac{2\pi}{M}; r = 0, 1, \ldots, M-1$. To show this, denote the z

transform of x evaluated on the unit circle by $Z_\theta[x]$, where the transform and inverse transform relations are:

$$Z_\theta[x] = \sum_{k=-\infty}^{\infty} x(k)e^{-i\theta k};\ \theta \in [0, 2\pi] \tag{4.3-4}$$

$$x(k) = \frac{1}{2\pi}\int_0^{2\pi} d\theta e^{i\theta k}\ Z_\theta[x]$$

Then, the z-transform of the FFT of x is:

$$Z_\theta\left[\mathfrak{F}_r[x(k)]\right] = W_M(\theta - \theta_r)Z_\theta[x],\ \theta_r \triangleq r\frac{2\pi}{M} \tag{4.3-5}$$

$$W_M(\xi) = \frac{1 - e^{-iM\xi}}{M(1 - e^{-i\xi})}$$

$W_M(\xi)$ is unity for $\xi = 0, 2\pi$ and vanishes at $\xi = n\frac{2\pi}{M}; n = 1, 2, \ldots M-1$. Hence, at $\theta = r\frac{2\pi}{M}$, (4.3-5) gives $Z_\theta\left[\mathfrak{F}_r[x(k)]\right] = Z_\theta[x]$. Now for a linear time invariant system:

$$Z_\theta[y] = H(\theta)Z_\theta[x] \tag{4.3-6.a}$$

But, taking the z transform of (4.3-3.b) and using (4.3-5) gives:

$$Z_\theta[y] = \sum_{r=0}^{M-1} H_r\ W_M(\theta - \theta_r)\ Z_\theta[x] \tag{4.3-6.b}$$

Comparing the above expressions and using the properties of $W_M(\theta - \theta_r)$ we see that:

$$H_r = H\left(\theta = \theta_r = r\frac{2\pi}{M}\right) \tag{4.3-7}$$

$$r = 0, 1, \cdots, M-1$$

Hence, a replicator based upon (4.3-3.b) is, in essence, a frequency analyzer. (4.3-3.b) represents the dynamic system by an interpolation function approximation to its frequency response.

Referring again to Table 4-1, test ANC 5 was a checkout of the code for the FFT operation and the code needed to implement a basic replicator using the model (4.3-3.b). Following the code implementation and checkout, ANC 6 tested a SISO FFT replicator using the disturbance channel and one accelerometer on the MHPE tower. Clearly, the parameter M in (4.3-1) determines the number of frequency points at which the frequency response is matched. The expression for θ_r shows these points to be separated by $\frac{2\pi}{M}$ and this is the "frequency resolution" of the FFT-based replicator. In ANC 6, the frequency response was accurately matched by choosing M such that $\frac{1}{M}\Omega_s$ (Ω_s = sample rate in Hertz) is 1 Hz.

With good progress on ANC 6, we moved on to test a MISO (Multi-input, single output) FFT replicator and a simple form of controller in ANC 7. To describe the ANC 7 algorithms, Figure 4.3-1 first gives simplified block diagram conventions. The FFT block illustrated at the top of the figure computes the FFT of the input *x(k)* using the fast, simple recursion relation:

$$\mathcal{F}_r[x(k+1)] = e^{i\theta_r}\,\mathcal{F}_r[x(k)] + \frac{1}{M}[x(k+1) - x(k-M+1)] \tag{4.3-8}$$

$$\theta_r \triangleq r\frac{2\pi}{M}$$

which is readily obtained from (4.3-1). The output, $\bar{x}(k)$, contains the real and imaginary parts, given alternately, for all the FFT components.

The "divided" weights" device, shown in the lower half of Figure 4.3-1 synthesizes an FIR filter based on representation (4.3-3.b) where the frequency response values at the FFT frequency points (the *Hr's*) are computed from the quotient of two given frequency responses.

Obviously, both devices in Figure 4.3-1 are specialized types of linear filters and involve no adaptive adjustment of coefficients.

With the above conventions, Figure 4.3-2 shows the algorithms tested in ANC 7. The top half of the figure shows an FFT version of an FIR replicator. In the frequency-domain, this model is based on the system model:

$$Z_\theta[y] = G(\theta)Z_\theta[d] + P(\theta)Z_\theta[u] \tag{4.3-9}$$

y = sensor output

d = disturbance signal measurement

u = control command to actuator

$G(\theta)$ and $P(\theta)$ are termed, in the acoustics literature, the "primary path" and "secondary path" dynamics and we use the same terminology here. The replicator in Figure 4.3-2 attempts to approximate $Z_\theta[y]$ at the M FFT frequency points. The z-transform of $\hat{y}$ is given by:

$$Z_\theta[\hat{y}] = \sum_r G_r W_M(\theta - \theta_r) Z_\theta[d] + \sum_r P_r W_M(\theta - \theta_r) Z_\theta[u] \tag{4.3-10}$$

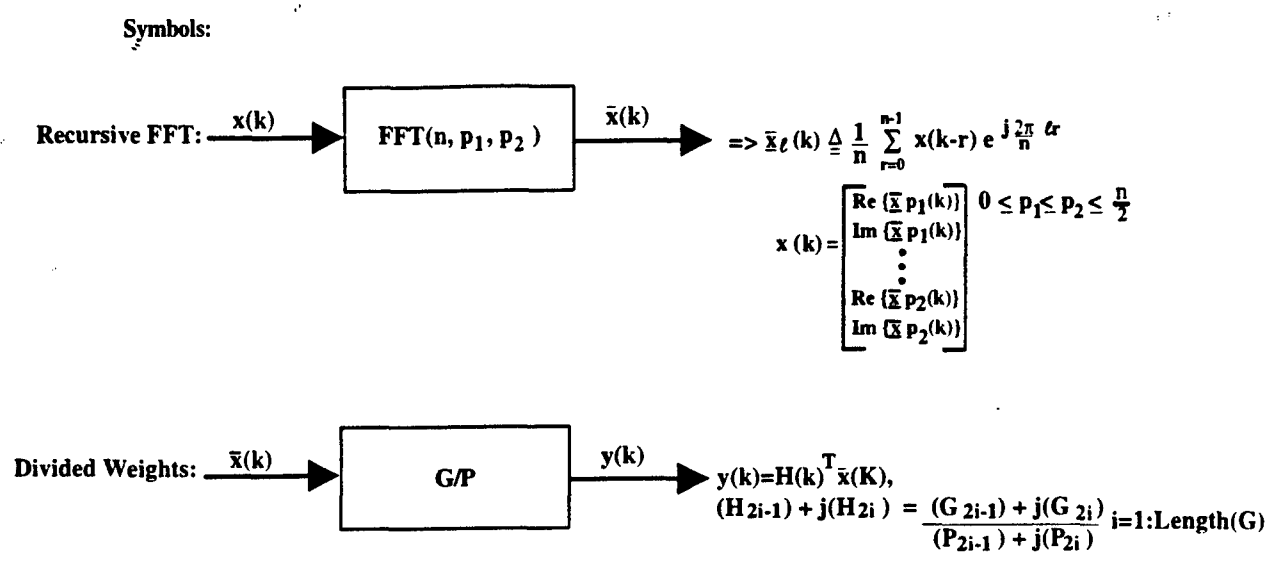
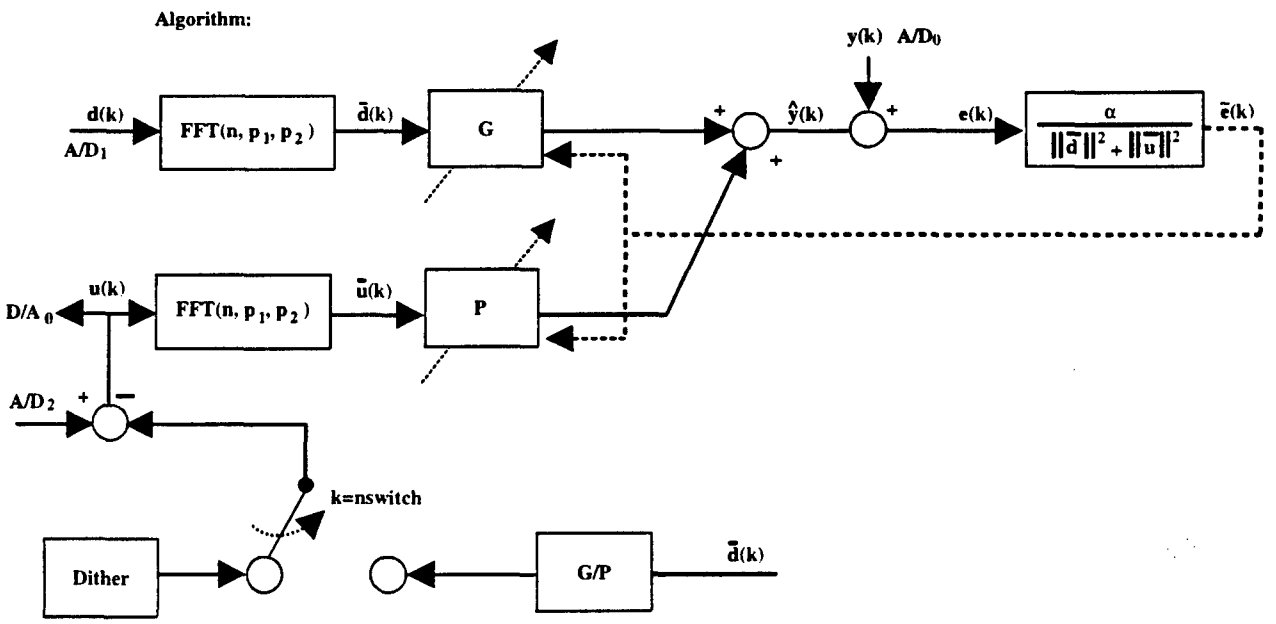


Figure 4.3-1: Some Block Diagram Conventions Relating to Frequency-Domain Neural Replicators



Results/Conclusions:

- f_{sample} = 500 Hz, n=500, p_1=0, p_2=40, nswitch/f_{sample} = 180 sec
- disturbance from 5-30 Hz
- large low frequency drifts observed when control switched on, which used up dynamic range

Figure 4.3-2: ANC 7, FFT Replicator for Plant ID, Control via Division of Identified Transfer Function Weights

Thus, if G_r and P_r converge to $G(\theta_r)$ and $P(\theta_r)$, respectively, then $Z_\theta[\hat{y}]$ exactly matches the system frequency response at the FFT frequency points. Once replicator convergence is obtained and one has a good frequency-domain model, the optimal feedforward controller is such that:

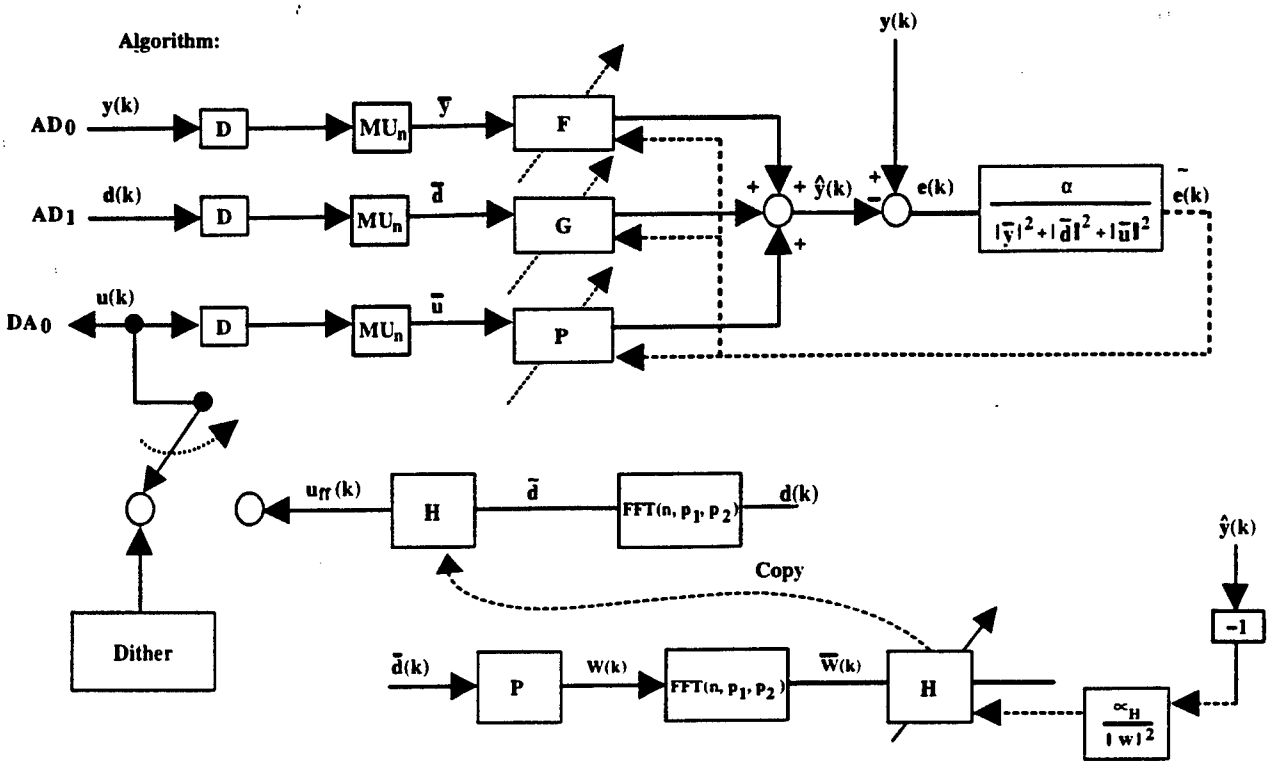
$$Z_\theta[u] = -\frac{G(\theta)}{P(\theta)} Z_\theta[d] \tag{4.3-11}$$

The implementation of this is shown in the lower half of Figure 4.3-2, using the "divided weights" filter defined in Figure 4.3-1. With the values of the parameters indicated at the bottom of Figure 4.3-2, persistent errors and drift were observed in the low frequency weights (<5 Hz) of the replicator and, in consequence, large drifts were obtained in the control input.

At first, it was thought that the above might be due to the use of the "divided weights" device for computing the control and due to the representation of the IIR by an FIR replicator. Hence the algorithms of ANC 8, shown in Figure 4.3-3, were tried. In this case, the replicator is an ARMA model and the controller is an FFT version of an IIR replicator, which computes the needed cancelling control input given the identified secondary path dynamics, P.

Basically the same drift in the low frequency weights was observed in ANC 8 as in ANC 7. It was concluded that the use of the recursive control computation, shown at the bottom of Figure 4.3-3, offered little benefit and that the use of an FIR versus IIR replicator was irrelevant to the difficulties experienced in ANC 7. In any case, a significant portion of the low frequency drift was traced to the round-off problems with ARMA replicators discovered in ANC 10.

Our next hypothesis was that the drift problems experienced in ANC 7 were due to the fact that replicators based on (4.3-3.b) are only " half frequency domain". Replicator inputs are FFT components but their outputs are still representations of the time domain output of the system being identified. Implicit in this setup is that all FFT components and corresponding frequency-domain weights have to be employed—including the lowest frequency components. As the low frequency components are most subject to measurement errors and drift and are associated with the slowest convergence rates anyway, it is, in retrospect, no surprise that the replicators could not approximate the low frequency behavior within reasonable learning periods. However, to achieve good feedforward control over the frequency *band* of the disturbances, it is not actually necessary to approximate the system transfer functions everywhere in frequency but only in the disturbance band, starting at some reasonable low frequency cutoff. Thus, our response to the difficulties of ANC 7 was to devise replicators that only operate on a designated frequency band—thus eliminating the processing of very low frequency components. To do this, it is necessary to use fully frequency domain replicators for which the inputs are FFT components *and* the output is a representation of the system *frequency* response (not its time-domain response).



Results:

- **unstable learning, but I.D'd plant seemed good. (!?)**
- **f_S = 500 Hz, n=15, N=500, d=0.01 $\propto_H$ = 0.01, d(k) 5-30 Hz random 120 seconds of learning**

Figure 4.3-3: ANC 8 2x1 Series Parallel I.D. XLMS FFT Weight Adjustment

In place of the model (4.3-3.b) suppose we approximate $\mathcal{F}_r[y(k)] \simeq \hat{H}_{\theta r}\, \mathcal{F}_r[x(k)]$—i.e. attempt to replicate an FFT component of *y(k)* rather than *y(k)* itself. The ANC replicator corresponding to this setup is shown in Figure 4.3-4.a, where $< \cdot >$ denotes the time average:

$$< x(k) > = \frac{1}{T} \sum_{m=0}^{T-1} x(k-m) \tag{4.3-12}$$

and we assume that the actual system response is given by (4.3-6.a)

Following the rules of ANC construction, the update rule for $\hat{H}_{\theta r}$ in Figure 4.3-4a is given by:

$$\hat{H}_{\theta r}(n+1) = \hat{H}_{\theta r}(n) + \frac{\alpha}{\left\langle \left| \mathcal{F}_r[u] \right|^2 \right\rangle} \left\langle \mathcal{F}_r[y]\, \mathcal{F}_r^*[u] - \hat{H}_{\theta r} \left| \mathcal{F}_r[y] \right|^2 \right\rangle \tag{4.3-13}$$

and for $\alpha << 1$ we may use the averaging approximation:

$$\hat{H}_{\theta r}(n+1) \simeq \hat{H}_{\theta r}(n) + \frac{\alpha}{\left\langle \left| \mathcal{F}_r[u] \right|^2 \right\rangle} \left\langle \mathcal{F}_r[y] \mathcal{F}_r^*[u] - \hat{H}_{\theta r} \left| \mathcal{F}_r[y] \right|^2 \right\rangle \tag{4.3-14}$$

$$= \hat{H}_{\theta r}(n) + \alpha \left[\frac{\left\langle \mathcal{F}_r[y] \mathcal{F}_r^*[u] \right\rangle}{\left\langle \left| \mathcal{F}_r[y] \right| \right\rangle^2} - \hat{H}_{\theta r}(n) \right]$$

Now consider the quotient within brackets. By Parseval's Theorem and (4.3-5):

$$\lim_{T \to \infty} \left\langle \mathcal{F}_r[y] \mathcal{F}_r^*[u] \right\rangle = \frac{1}{2\pi} \int_0^{2\pi} d\theta \left| W_M(\theta - \theta_r) \right|^2 H(\theta) \left| Z_\theta[u]^2 \right| \tag{4.3-15}$$

Since $W_M\{\xi\}$ assumes nonnegligible values only near $\xi = 0$ and supposing adequate frequency resolution ($\frac{2\pi}{M}$ sufficiently small), we obtain:

$$\left\langle \mathcal{F}_r[y] \mathcal{F}_r^*[u] \right\rangle \simeq H_{(\theta r)} \left\langle \left| \mathcal{F}_r[u] \right|^2 \right\rangle \tag{4.3-16}$$

Thus the update equation becomes:

$$\hat{H}_{\theta r}(n+1) = \hat{H}_{\theta r}(n) + \alpha \left(H(\theta_r) - \hat{H}_{\theta r}(n) \right) \tag{4.3-17}$$

and with $\alpha < 2$, $\hat{H}_{\theta r}$ converges to $H(\theta_r)$. Thus the replicator of Figure 4.3-4a directly estimates the frequency response at one FFT frequency. Notice that the replicator operates on only one FFT component. As many or as few frequency components as are desired can be processed in parallel by a multiplicity of such mutually independent replicators. Besides rapid convergence and simplicity, this scheme allows one to identify systems over a specified frequency window and exclude unnecessary low frequency components.

Figure 4.3-4b shows the forward path signal flow for combining several "frequency bucket" replicators, and effecting the inverse FFT to synthesize a time-domain output. This is needed later to synthesize the proper control inputs. As a preliminary step, the forward path algorithm of the "FFT weight filter" in Figure 4.3-4b was implemented and successfully tested in ANC 9.

Subsequently, ANC 11 tested the capability to identify the disturbance to error sensor transfer function value at an FFT frequency point (in a single "frequency bucket") preparatory to a complete identification exercise in ANC 12.

For greater stability and accuracy in ANC 11 we actually employed a "batch" version of the algorithm in Figure 4.3-4a. Figure 4.3-5 shows the system tested in ANC 11. The algorithm is depicted in previously established ANC block-diagram notation and is devoted to identifying the transfer function at a single frequency point. Identification over a frequency band is

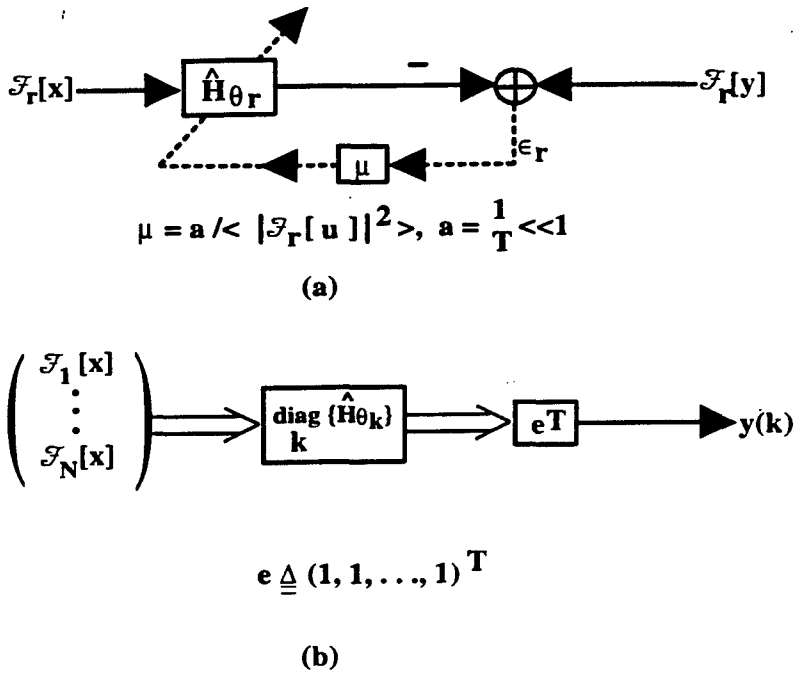
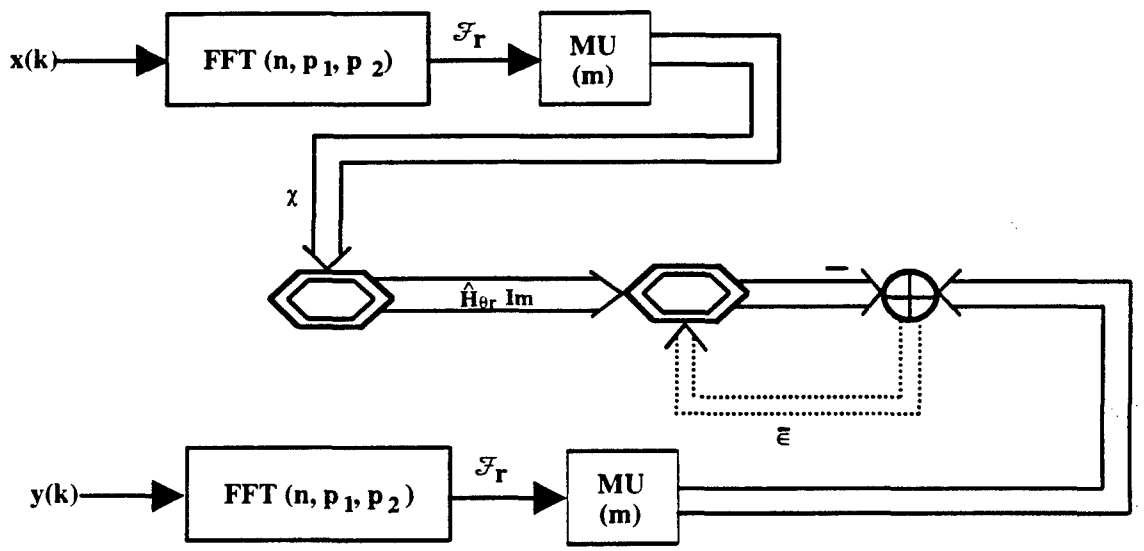


Figure 4.3-4: Devices for Fully Frequency-Domain ANC Components. (a) Single Channel Replicator and (b) Forward Path Dynamics Combining all Channels to Produce a Time-Domain Output



Note: $\Delta \hat{H}_{\theta r} = \mu \chi^H \bar{\epsilon},\ \mu = \alpha / \|\chi\|^2,\ \alpha \leq 2$

Exit if: $\|\bar{\epsilon}\|^2 / \left((m-1)\|x\|^2\right) < \lambda^2$ **(= required mean-square accuracy of** $\hat{H}_{\theta r}$**)**

Figure 4.3-5: Batch Version of Fully Frequency-Domain ANC Replicator Tested in ANC 11

accomplished by an array of modules of the type shown, all running in parallel. The basic inputs to the replicator assigned to identify at frequency θ_r are the FFT components of the input and output; $\mathcal{F}_r[x(k)]$ and $\mathcal{F}_r[y(k)]$, respectively. The input and output ganglia are connected by a synaptic bundle with weight matrix constrained to be proportional to the identity matrix. Because of this constraint, the weight increment is proportional to the inner product $x^H \overline{\epsilon}$ as shown in the formula in Figure 4.3-5. This frequency-domain identifier basically time averages the results that would be given by the algorithm of Figure 4.3-4a, thereby achieving good noise filtration properties. The number of time samples used in the averaging is given by the parameter m. Assuming the learning rate constraint, α, is less than 2, $\hat{H}_{\theta_r}$ will converge to $H(\theta = \theta_r)$ supposing that x and y are related as in (4.3-6.a)). In the subsequent work, α=1.0 was used throughout. Because the weight matrix is diagonal, the computational burden is actually rather small and, as will be seen below, a large number of frequency channels of the form of Figure 4.3-5 can be implemented simultaneously on the DSP card. Finally, as indicated at the bottom of Figure 4.3-5, we use x and $\overline{\epsilon}$ to compute an estimate of the mean square identification error. This is compared with λ^2, the user-supplied requirement for mean square accuracy. If this test is passed, the algorithm terminates and announces the determination of a frequency response estimate of the desired accuracy.

The above algorithm was tested in ANC 11 and the accuracy of the results was found to be very good when compared with the results of the spectrum analyzer. The algorithm also showed its ability to monitor its own accuracy. With this encouragement, ANC 12 went on to demonstrate complete identification over a frequency band. In ANC 12, the algorithm of Figure 4.3-5 is simply repeated for every FFT frequency point within the desired frequency band, and the results for $\hat{H}_{\theta 1}, \ldots, \hat{H}_{\theta n}$ are assembled to form the FIR filter representing the identified system.

As noted in Table 4-1, results for ANC 12 were quite good. For example, using 500 delay stages in the FFT operation with a sample rate of 250 Hz (giving a 0.5 Hz frequency resolution), the transfer function from the disturbance command to one of the MHPE tower accelerometers was identified over the 5 to 25 Hz band with 51 "frequency buckets."

Figure 4.3-6a, b compares the transfer function gain and phase plots of the identified filter (dashed line) with FRF data (solid lines). The match is excellent for both gain and phase. In neither ANC 12 nor in any subsequent tests were roundoff error problems observed.

With successful performance of the frequency-domain system identifiers, we proceeded to implement the frequency-domain control algorithms. Figure 4.3-7 shows the overall controller structure in which each frequency channel is identified and controlled separately and in parallel with all other channels. First, y and d are subjected to FFT operations. The FFT blocks in Figure 4.3-7 show the digital filter realization of the recurrence relation (4.3-8). The

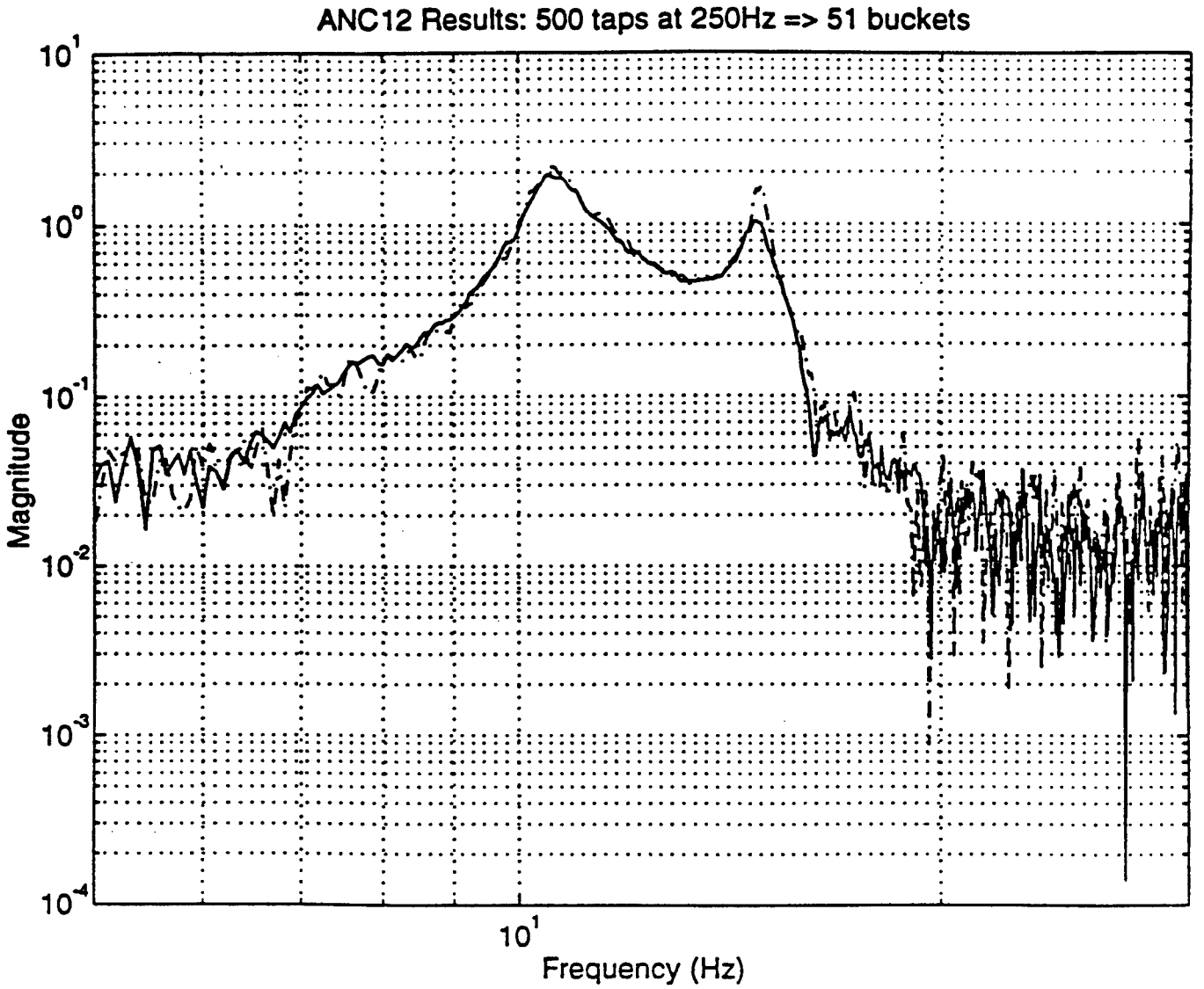


Figure 4.3-6a: Gain of Disturbance to Sensor Transfer Function. Frequency-Domain ANC Replicator (dashed line) is Compared with FRF Data (solid line)

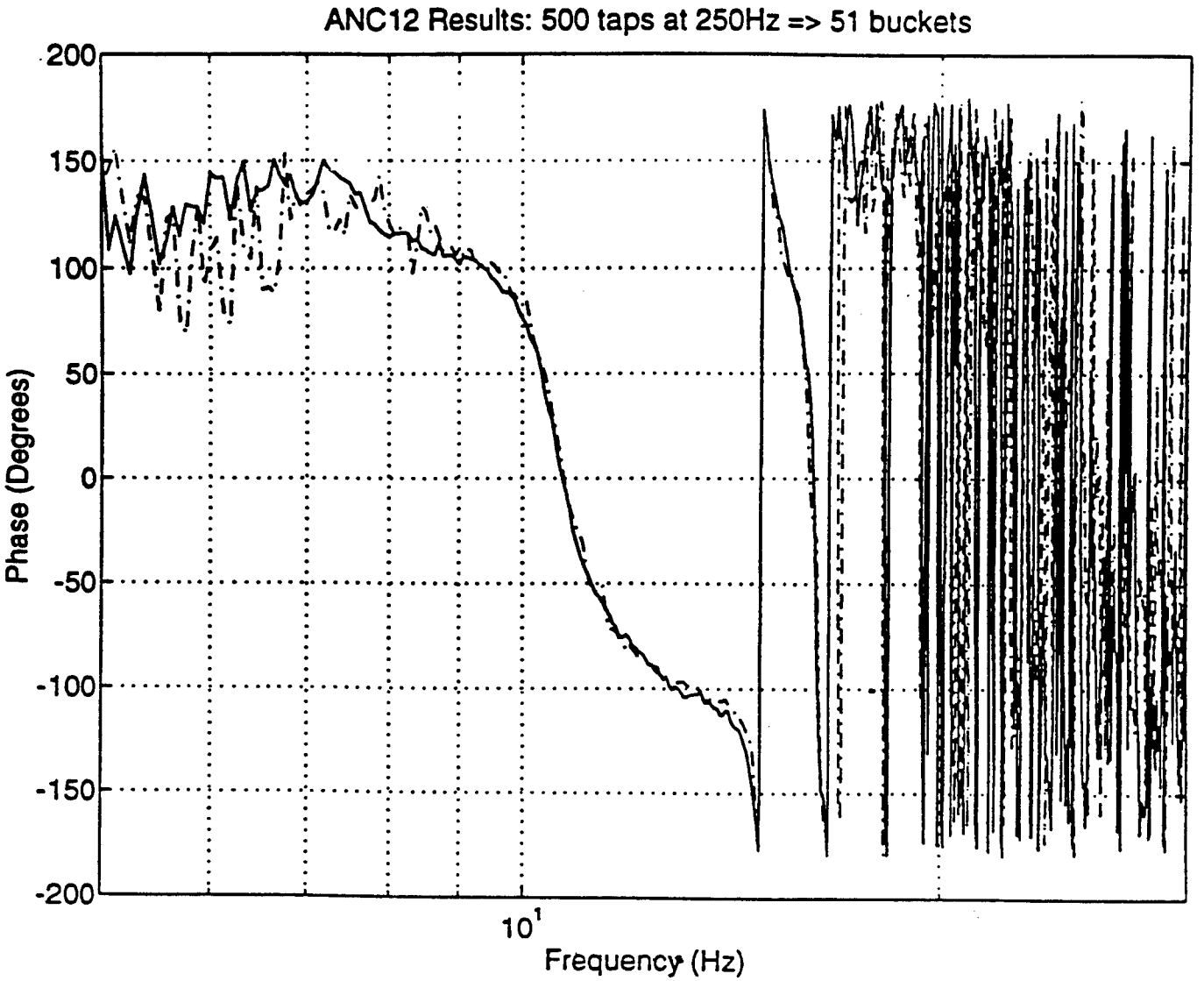


Figure 4.3-6b: Same Comparison as in Figure 4-6a Except that the Transfer Function Phase is Shown for ANC Replicator (dashed line) and FRF Data (solid line)

r^{th} FFT components of y and d are denoted by $\hat{y}_r$ and $\hat{d}_r$, respectively. Each pair of FFT components having the same index (the same center frequency) are handled separately. For $r = 1, \ldots, M$, each pair $\left(\hat{y}_r, \hat{d}_r\right)$ are input to the *rth* channel subsystem consisting of an *rth* channel identifier in series with an *rth* channel control adaptor. The channel identifier outputs an estimate, $\hat{\overline{G}}_r$ of $\overline{G}_r\,(\theta = \theta r)$ where $\overline{G}$ is the closed-loop transfer function matrix from d to y and also passes along $\hat{d}_r$. With the inputs $\hat{d}_r$ and $\hat{\overline{G}}_r$, the channel r control adaptor outputs its component, $\hat{u}_r$ of the actuator drive signal vector. Finally, outputs of all the channel control adaptors are summed to form the total actuator command signal vector, $u(n)$.

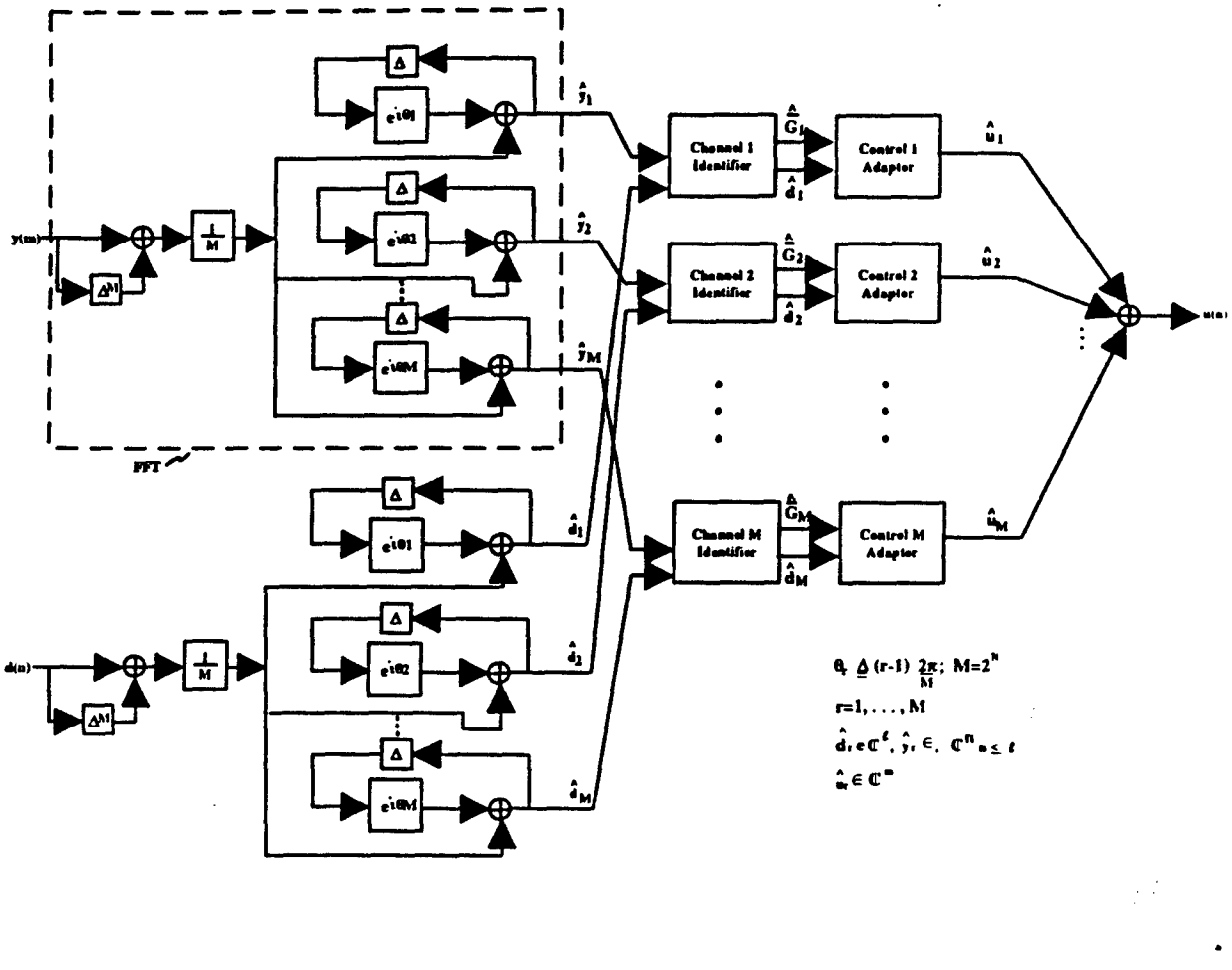


Figure 4.3-7: Overall Controller Processes the Separate Frequency Channels in Parallel

To understand the details of the identifier and control adaptor devices in Figure 4.3-7 we sketch the control issues that must be addressed. First, repeating (4.3-9), the system being controlled has the frequency-domain form:

$$Z_\theta[y] = G(\theta)Z_\theta[d] + P(\theta)Z_\theta[u] \tag{4.3-18}$$

$$y \in \mathrm{IR}^\ell, \quad u \in \mathrm{IR}^m$$

The feedforward controller is a linear filter with input d of the form of (4.3-3.b):

$$u(n) = \sum_{k=0}^{M-1} H_k \, \mathcal{J}_k[d(n)] \tag{4.3-19}$$

where $H_k \, \mathcal{J}_k[d(n)]$ corresponds to $\hat{u}_k$ in Figure (4.3-7). Substitution of (4.3-19) into (4.3-18) and use of (4.3-5) gives the closed-loop relation:

$$Z_\theta[y] = \overline{G}(\theta)Z_\theta[d]$$

$$\overline{G}(\theta) = G(\theta) + P(\theta)\sum_{k=0}^{M-1} H_k \, W_M(\theta - \theta_k) \tag{4.3-20}$$

so that $\overline{G}$ defines the closed-loop transfer function matrix. From the above, it is seen that the basic control strategy is to (somehow) identify the quantity $P^{-1}(\theta_r)G(\theta_r)$ for all θr in the desired control band, then set $H_r = -P^{-1}(\theta_r)G(\theta_r)$. This makes $\overline{G}(\theta)$ pass through zero at each FFT frequency point. With adequate resolution, this strategy produces the desired level of response suppression over the required frequency band.

We desire to carry out the above control strategy so that the identifications are performed and the cancelling control is synthesized *simultaneously* without the use of extraneous test signals (e.g. broadband dither introduced into u to identify $P(\theta)$) and without interruption of normal operation. There are two basic approaches:

(A) Using only $\hat{d}_k$, $\hat{u}_k$ and $\hat{y}_k$, identify $G(\theta_k)$ and $P(\theta_k)$ separately, then generate $H_k = -P^{-1}(\theta_k)G(\theta_k)$

or:

(B) Identify $\mathcal{Q}_k \triangleq P^{-1}(\theta_k)$ and $\overline{G}(\theta_k)$ and then generate H_k from:

$$H_k(n+1) = H_k(n) - \mathcal{Q}_k\overline{G}_k(n) \tag{4.3-21}$$

The recursive scheme (4.3-21), in effect, identifies $-P^{-1}(\theta_k)G(\theta_k)$ to produce the desired controller. To see this, use the expression for $\overline{G}$ in (4.3-20):

$$
\begin{aligned}
H_k(n+1) &= H_k(n) - \mathcal{Q}_k\left[G(\theta_k) + P(\theta_k)H_k(n)\right] \\
&= H_k(n) - \left[P^{-1}(\theta_k)G(\theta_k) + H_k(n)\right] \quad \left(\text{because } \mathcal{Q}_k = P^{-1}(\theta_k)\right) \\
&= -P(\theta_k)G(\theta_k)
\end{aligned}
$$

Approach A has had considerable success, especially in tonal cancellation. However this scheme has a "slow mode" of convergence that, over extended operation, progressively slows down the convergence rate. On the other hand, approach B does not exhibit this behavior and is very robust because (4.3-21) can converge to the cancelling controller even when the error $\left\|\mathcal{Q}_k - P^{-1}(\theta_k)\right\|$ is quite large. Thus, approach B was chosen for the present study.

The basic, preliminary identification task needed to implement approach B is the identification of $\overline{G}(\theta_r)$, $r = 1, \ldots, M$. Each of the $\overline{G}(\theta_r)$ is determined by one of the channel identifiers in Figure 4.3-7. The channel r identifier has the form shown in Figure 4.3-5 with $\hat{d}_r$ and $\hat{y}_r$ as inputs and $\hat{G}_r$ as the output estimate of $\overline{G}(\theta_r)$. For this "batch" frequency-domain identifier, it is most convenient to run the algorithm in separate time blocks, each of length M. In other words, the estimate $\hat{\overline{G}}_r$ is held constant for M time steps after which a new estimate is produced. If M is roughly the correlation time of each FFT coefficient, the algorithm of Figure 4.3-5 takes L steps to arrive at a converged estimate where L is typically 3 to 5. With this time-block mode of operation, the control adaptors update the H_r's only once every "identification epoch" of LM time steps. Thus, in describing the control adaptor operation, the time index "n" is used to characterize the values of $\mathcal{Q}_k, H_k$ and $\hat{\overline{G}}_k$ on the n^{th} identification epoch.

Assuming the k^{th} channel identifier has arrived at an acceptable estimate of $\overline{G}(\theta_k)$ at time "n", we have from (4.3-20):

$$\hat{\overline{G}}_k(n) = G(\theta_k) + P(\theta_k)H_k(n) + \eta_k(n) \tag{4.3-22}$$

where η_k represents sensor errors, instrumentation noise and identification error. A bound on the noise fluctuation can be assumed in the form:

$$\max_n \left\|\eta(n) - \eta(n-1)\right\| \le \sigma \tag{4.3-23}$$

Since η_k is largely the consequence of the electronics hardware being used, we may estimate (on the basis of test results) a value of σ. This was the procedure used in early tests; subsequently the algorithms were augmented by an on-line estimation of σ. Also, for future reference, define the notation:

$$\Delta \bullet(n) \triangleq \bullet(n) - \bullet(n-1) \tag{4.3-24}$$

and note from (4.3-22) that:

$$\Delta\, \hat{\overline{G}}_k(n) = P(\theta_k)\,\Delta\, H_k + \Delta\eta_k(n) \tag{4.3-25}$$

With the above relations, we can describe the operation of the k^{th} channel control adaptor, which is shown in Figure 4.3-8. We use the established ANC block diagram conventions. The main

input is $\hat{\bar{G}}_k(n)$. Δ^{ML} means a time delay back to the previous identification period. The lower portion of Figure 4.3-8 is a device that makes use of the correlation between fluctuations of $\hat{\bar{G}}_k$ and fluctuations of H_k to determine $P^{-1}(\theta_k)$. Essentially, this is an identifier for a linear mapping from ΔH_k to $\Delta \hat{\bar{G}}_k(n)$. From (4.3-25) it can be seen that $\mathcal{L}_k$ converges to $P^{-1}(\theta_k)$ (since, ignoring noise, $\Delta H_k = P^{-1}(\theta_k)\Delta \hat{\bar{G}}_k$). Given $\hat{\bar{G}}_k$ and a copy of $\mathcal{L}_k$, the upper half of Figure 4.3-8 implements (4.3-21). Specifically, following the block diagram conventions; the H_k update is:

$$H_k(n+1) = H_k(n) - \mathcal{L}_k(n+1)\hat{\bar{G}}_k(n)$$

—i.e. H_k is updated just after $\mathcal{L}_k$ is. Note that the learning rate for the $\mathcal{L}_k$ adaptation is adjusted during operation according to the formulae as given at the bottom of Figure 4.3-8. This prevents undue noise sensitivity when convergence is nearly complete and $\overline{G}$ is small. Here, σ is the bound on noise fluctuation, (4.3-23) and ϵ is a prespecified tolerance on the $P^{-1}(\theta_k)$ identification, i.e.:

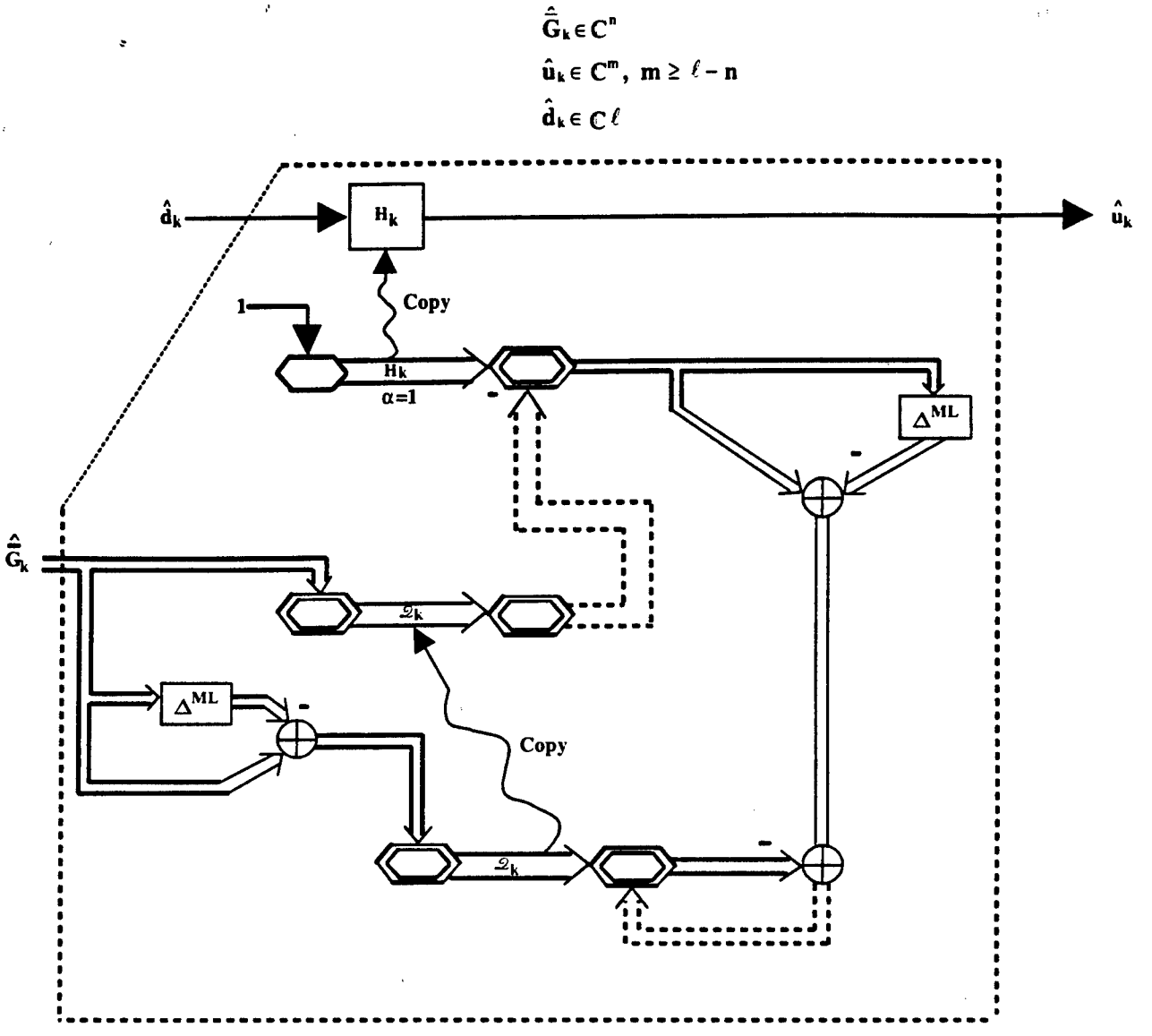
$$\epsilon \;=\; \text{required upper bound on } \left\| I - P(\theta_k)\mathcal{L}_k(n) \right\| \qquad (4.3\text{-}26)$$

For the above algorithm, we can state the following result:

> Suppose $d(k)$ is a stationary random time series and $P(z)$ is stable, strictly proper and minimum phase. Then for the algorithm of Figure 4.3-8 and for any initial values of $\mathcal{L}_k$ and H_k:
>
> $$\lim_{n \to \infty} \left\| I - P(\theta_k)\,\mathcal{L}_k(n) \right\| \le \epsilon \qquad \text{a. (4.3-27)}$$
>
> $$\lim_{n \to \infty} \left\| \hat{\bar{G}}_k(n) \right\| < \sigma / (1 - \epsilon) \qquad \text{b. (4.3-27)}$$
>
> with probability one.

Thus the algorithm is guaranteed to achieve the desired accuracy in identifying the "inverse secondary path", $P^{-1}(\theta_k)$ and in reducing the closed-loop transfer function down to the instrumentation noise floor.

The identification and control algorithms described above were implemented in ANC 12. This test sequence concentrated on a single control loop involving one of the MHPE tower-mounted LPACTs and its colocated accelerometer. Some six tests were performed and the results were generally quite good. For example, Figure 4.3-9 shows one case in which the desired control range is 5 to 20 Hz and the frequency resolution (interval between FFT points) is 0.5 Hz with a sample rate of 250 Hz. It is seen that the dominant group of modes near 10 Hz is reduced by approximately 20 dB. The result shown in Figure 4.3-9 was obtained after 3 iterations of the



Channel K Control Adaptor

In the $\mathcal{Q}_k$ Adaptation, the Learning Rate is:

$$\alpha = \begin{cases} 1, \left|\Delta \hat{\bar{G}}_k(n)\right| > \sigma / \epsilon, \epsilon < 1 \\ 0, \text{ otherwise} \end{cases}$$

σ = bound on noise fluctuation

ϵ = ID error tolerance

Figure 4.3-8: Selected Structure for the k^{th} Channel Control Adaptor is Essentially an Identifier for $P^{-1}(\theta_k)$ and $P^{-1}(\theta_k)G(\theta_k)$

algorithm in Figure 4.3-8. In this scheme, most of the computation time is taken up by the identification process since the length of each identification epoch must be more than the

correlation time of each FFT component of y or d. Since the correlation time is roughly the reciprocal of the frequency resolution the overall convergence time is inversely proportional to frequency resolution.

Despite favorable results on ANC 12, the closed-loop response after convergence was found to vary noticeably from run to run. After further analysis this was traced to two causes. First on each epoch, the identification algorithm in Figure 4.3-5 was run for a prespecified time interval and a fixed number of epochs was used for each control update. Thus the identifier provided an estimate of $\hat{\overline{G}}_k$ to the controller without regard to the actual accuracy of the result. Secondly, in the above scheme the controller adjusts its gain for the same number of FFT components as are included in the estimator—i.e. the frequency resolutions of the identifier and controller are the same. In this case, there is discernible interaction among the frequency channels, particularly where $\overline{G}(\theta)$ varies rapidly over a frequency bucket. Such interactions cause nonrepeatable variations in the final controller. To reduce these interactions we observe that if the plant is adequately approximated by an FIR filter of M delays and the controller is also an FIR filter of the same order then the identifier model of the close-loop must have $2M$ delays—i.e. the frequency resolution of the identifier ought to be twice that of the controller. When this condition imposed, it is clear that cross-channel interactions are minimized.

Therefore in ANC 13, we modified the algorithm of ANC 12 by (1) running the identifier so that it terminates only when the mean-square error estimate falls below λ^2 and (2) setting up the controller so that it addresses every other FFT point that is processed by the identifier. Modification (1) ensures estimates of $\overline{G}(\theta)$ of uniform quality. Modification (2) gives the identifier twice the frequency resolution as the controller. With these improvements, ANC 13 converged somewhat more quickly and far more reliability than did ANC 12. The algorithm development would have concluded here if it were not for one fundamental difficulty.

The problem was that because each frequency channel was treated as if it were independent and the fully frequency-domain ANC replicator (Figure 4.3-5) was used, the ANC 13 algorithm tended to control the system at the centers of the frequency channel but not elsewhere. What often occurred was that deep (>20 dB) nulls were created at the center of the various channels, but elsewhere the closed-loop response was near the open-loop levels.

One of the essential difficulties was traced to the fact that the fully frequency domain replicator, Figures 4.3-4;5, when used to identify the closed-loop transfer function $\overline{G}(\theta)$ in equation (4.3-20), actually introduces coupling among the several frequency channels. Recall that the replicator works by correlating $\mathcal{F}_r[y]$ with $\mathcal{F}_r[d]$. Using (4.3-5) and (4.3-20), we find:

$$Z_\theta\left[\mathcal{F}_r[y]\right] = W_{2M}(\theta-\theta_r)\left[G(\theta) + P(\theta)\sum_{k=0}^{M-1} H_k W_M(\theta-\theta_k)\right] Z_\theta[d] \qquad (4.3\text{-}28)$$

From this, it is evident that $\mathcal{F}_r[y]$ has significant frequency content outside the r^{th} frequency channel coming from the k^{th} ($k \neq r$) control channel input. This is still a source of significant variability in the identification and control results.

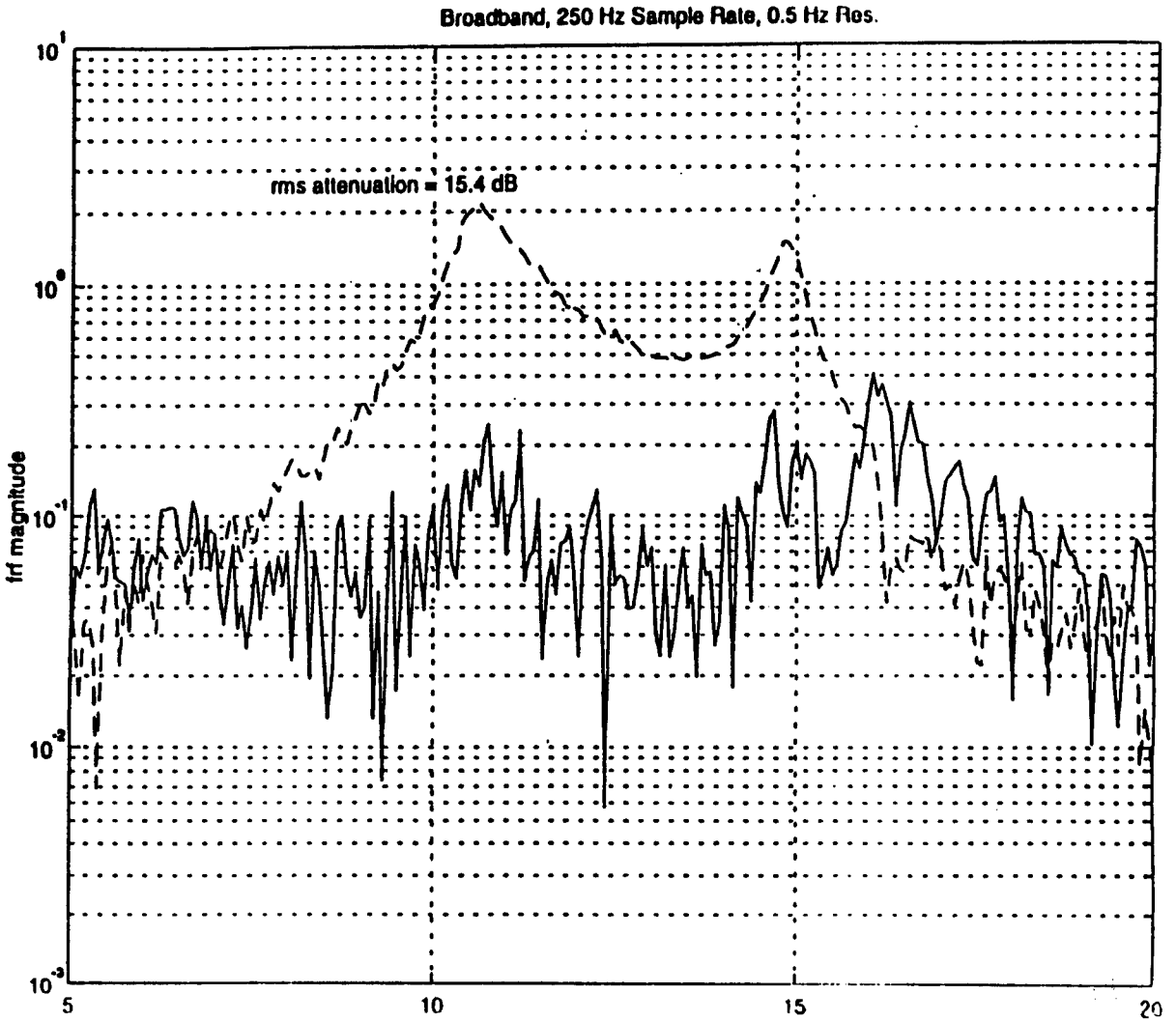


Figure 4.3-9: ANC 12, Example Case: Open and Closed-Loop Results for the Gain of the Transfer Function from Disturbance to Error Sensor

A second and more significant difficulty is that the frequency domain replicator and independent channel controllers take account of the closed-loop response only at the channel center frequencies. The algorithm has no mechanism to recognize closed-loop response in between the center frequencies. This primarily accounts for the qualitative results noted above for ANC 13.

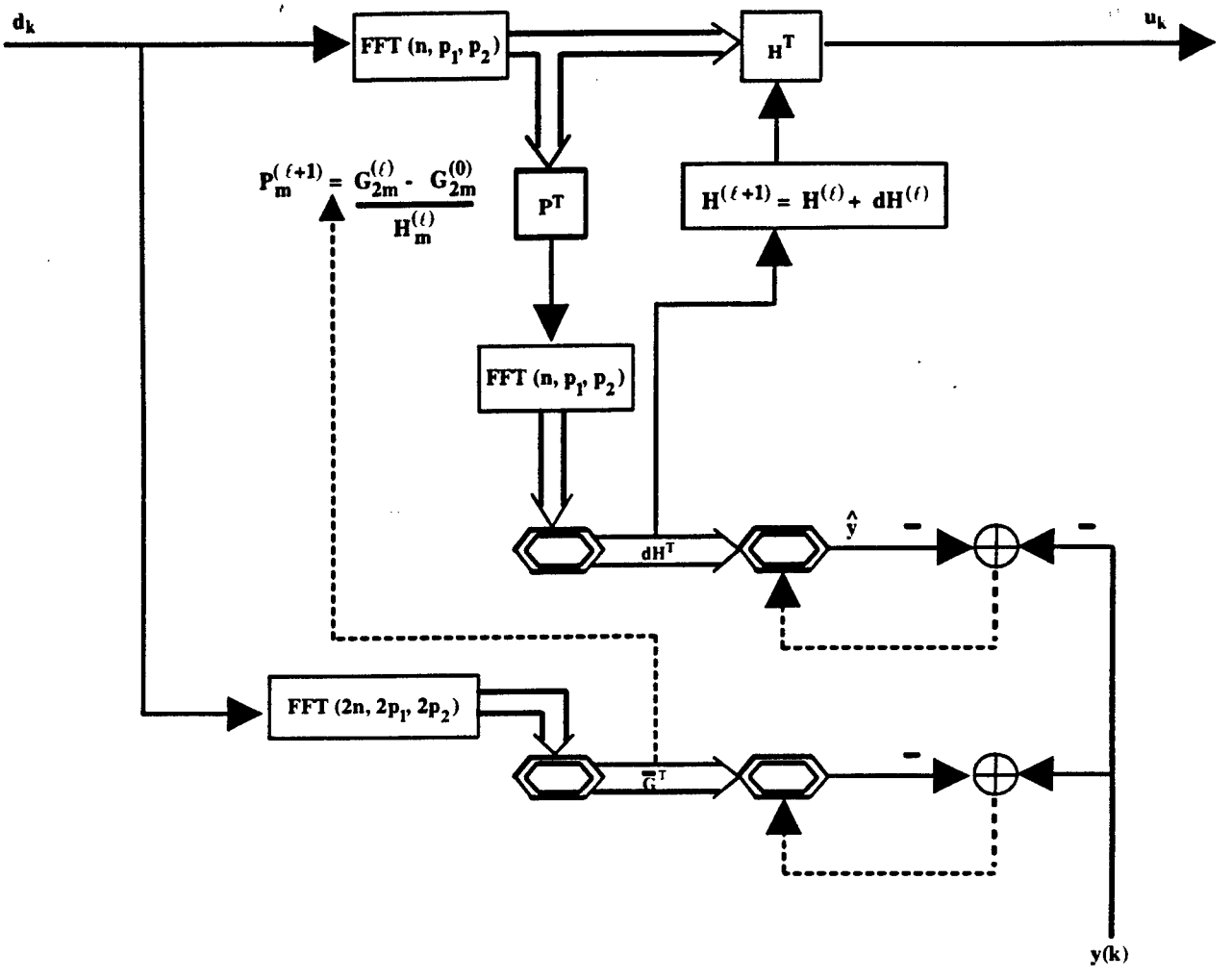
Upon reconsideration of the various replicator forms tested, it was decided to use the partial frequency domain versions of ANC of the type shown in Figure 4.3-2. These are based on the model form (4.3-3.b) and employ FFT inputs but time domain outputs. Figure 4.3-10 shows the mixed time-and frequency-domain algorithm used for ANC 14. This constitutes the finally successful algorithm adapted as the basis for our final MHPE demonstrations.

The algorithm shown in Figure 4.3-10 simultaneously identifies and controls the system. As in ANC 12 and 13, closed-loop system identification and control adaptation occur in epochs which are set to a fixed period (4 minutes in most experiments). $\overline{G}(\theta)$ is identified (see the bottom of Figure 4.3-10) using a model of the form (4.3-3.b). The input is the 2n FFT of the disturbance measurement while the output is an attempted replica of the error measurement y. As discussed in connection with (4.3-3.b), when $\overline{G}$ converges, it converges to the closed-loop transfer function from d to y at the FFT frequency points. Note that the time-domain control signal, $u(k)$, is synthesized in a similar way—i.e. using an n point FFT of d. Thus, as in ANC 13 the frequency resolution of identification is twice that of control.

Using the difference of $\overline{G}$ over different identification epochs and the current value of the control gain vector, $H \in C^n$, the secondary path dynamics transfer function, P is synthesized. These computations rigorously produce the required secondary path response, assuming the closed-loop replicator converges perfectly. This approach is also relatively insensitive to noise due to the fact that the differences in $\overline{G}$ and the control transfer function (i.e. H) magnitudes are quite large compared to the noise, especially after several adaptations have occurred.

After subjecting the output of the P^T block to an n-point FFT operation, the result is input to another replicator with synaptic weight vector dH^T. This replicator matches its output to the time series signal $-y(k)$. In other words dH converges to the change in the control gain needed to make $dH(\theta)P(\theta)$ equal to $-\overline{G}(\theta)$. Thus, when dH is added to H, the desired feedforward controller is obtained. This is done during an identification epoch while maintaining the on-line gain, H, fixed. When identification epoch ℓ is complete, dH^{ℓ} is added to H^{ℓ} and H is updated. This approach to control update ensures that the *time* series $y(k)$ converges to zero in a least square error sense. Hence, the algorithm attempts to minimize the closed-loop frequency response magnitude both in the vicinity of the FFT points and between these points. Thus, the fundamental difficulty of ANC 13 is avoided. By using the model form (4.3-3.b), the algorithm also avoids the sensitivity to machine precision observed for ARMA models. Finally, we note that the controller gains are updated at the beginning of each epoch with a perturbation which is computed off-line. This guarantees that the control and identification processes are completely independent and produce well-defined estimates of the desired quantities.

The ANC 14 algorithm was the basis for the final experimental demonstration. We review the results achieved by this algorithm in the next section.



- **Superscript ℓ (e.g. H^{ℓ}) refers to learning epoch ℓ**
- **Learning for *dH* and $\overline{G}$ run simultaneously, and are reset for each epoch**
- **H is fixed during learning epochs**

Figure 4.3-10: Mixed Time-and Frequency-Domain Algorithm for ANC 14

4.4 Experimental Results for ANC 14 Algorithm-Final MHPE Demonstration

The final results for ANC continuous spectrum control are presented in this section. These experiments use the ANC 14 algorithm described above. The ANC 14 algorithm was run using several actuator and sensor configurations on the MHPE testbed. Results were found to be affected by linearity, signal-to-noise ratio, and the inherent limitations of the filters used to control the system.

All experiments were run with a 250 Hz digital sampling frequency and learning epochs of 4 minutes (240 seconds). In what follows, the parameters of interest are df (the control resolution), f_1 (the lowest control frequency), and f_2 (the highest control frequency). Note that cancellation can only be expected after three epochs (12 min) of computation, since the first epoch is used to

compute open-loop response, the second is used to compute the first secondary path estimate with an arbitrary control input, and the third is used to compute the first control perturbation. In one experimental configuration, the disturbance was injected into the D/A channel of LPACT #1 located on the MHPE tower while sensing and control were executed using LPACT #2 on the tower. For good signal-to-noise ratio, a disturbance level of 500 mV was used.

Results for the case of df=0.4, f_1 = 3.5 Hz and f_2=24 Hz are shown in Figures 4.4-1 through 4.4-3. Attenuation of 13.4 dB rms was obtained after 20 min of adaptation, with approximately 23.5 dB obtained on the major structural mode at around 16 Hz.

At this point, it was decided to try a direct comparison with the earlier ANC results. Thus, in the second test configuration investigated here, the disturbance was injected into an LPACT mounted on the main petal of the MHPE, as before on ANC 12. The disturbance LPACT had a much lower force output than the shaker, so an input level of 1 volt rms was used over the same 7.5-20 Hz range as previously. This still did not give a very high signal-to-noise ratio for identification, however. In order to remedy this deficiency, it was decided to increase the gain of the LPACT A/D output by a factor of ten. The final set of experiments was then run to establish the best achievable performance. The parameters were df=0.4, f_1=3.5 Hz and f_2=24 Hz. Figures 4.4-4 through 4.4-7 show the results, with perhaps the most impressive case being Figure 4.4-7, the result after 24 minutes of learning. Attenuation overall was 14.1 dB (probably more if noise spikes were averaged out), with approximately 23 dB attained on the primary resonance at 14.6 Hz.

In summary, the ANC 14 algorithm appears to attain reliable performance limited only by theoretical implementation considerations and the signal-to-noise ratio.

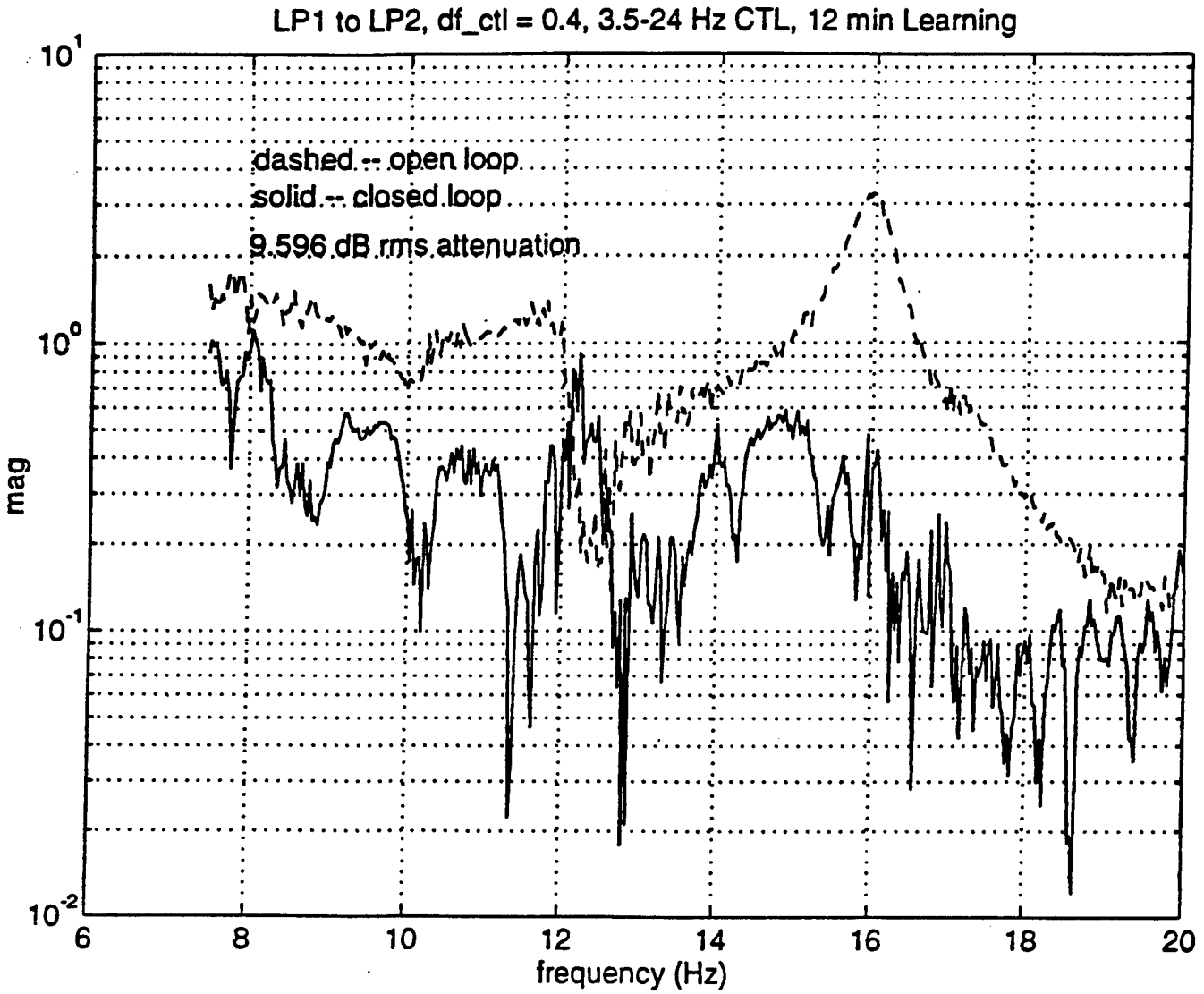


Figure 4.4-1: LPACT 1 disturbance, LPACT 2 control: Open-Versus Closed-Loop Response After 12 Minutes of Adaptation

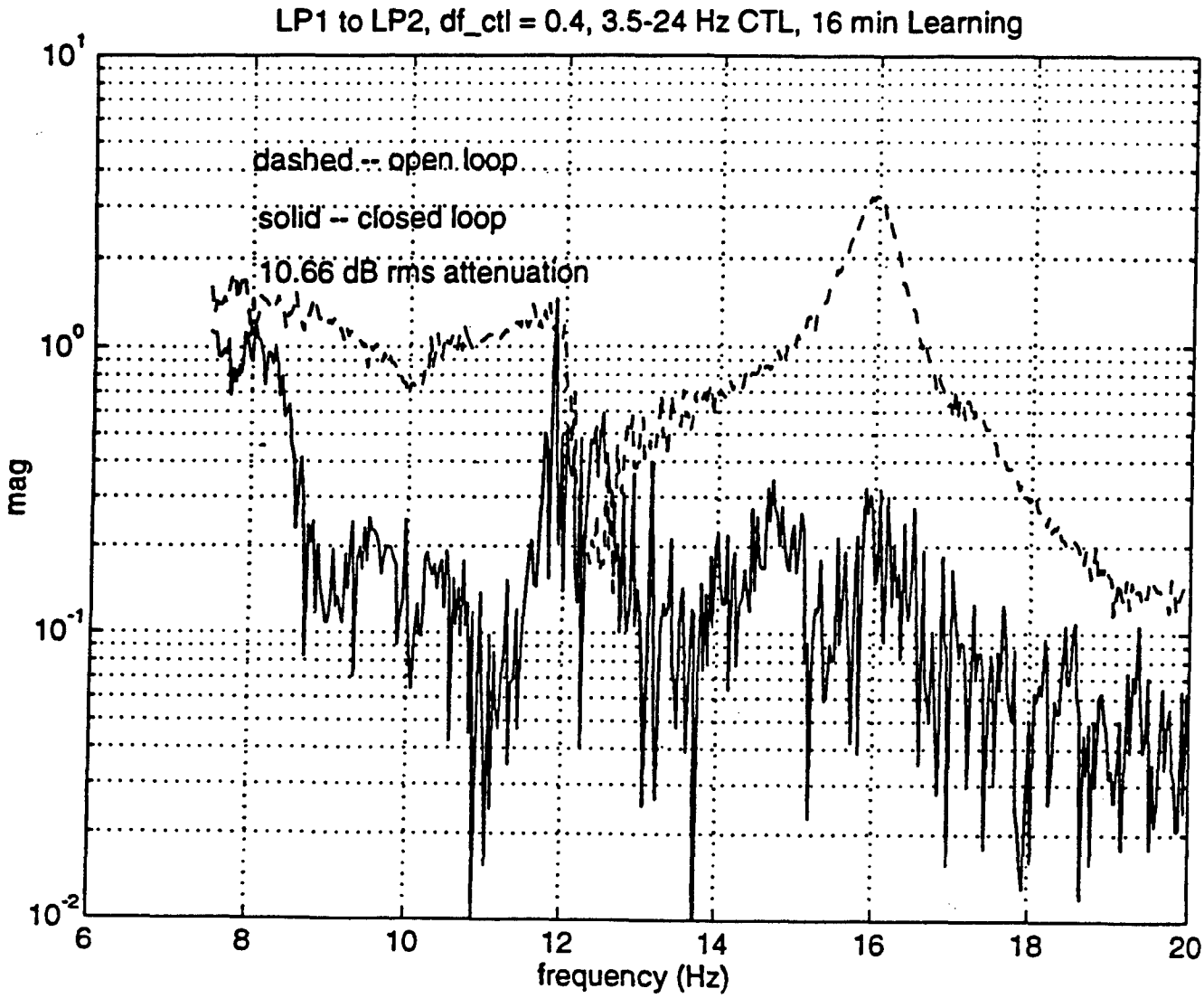


Figure 4.4-2: Same Case as Figure 4.4-1. Open-Versus Closed-Loop Response After 16 Minutes of Adaptation

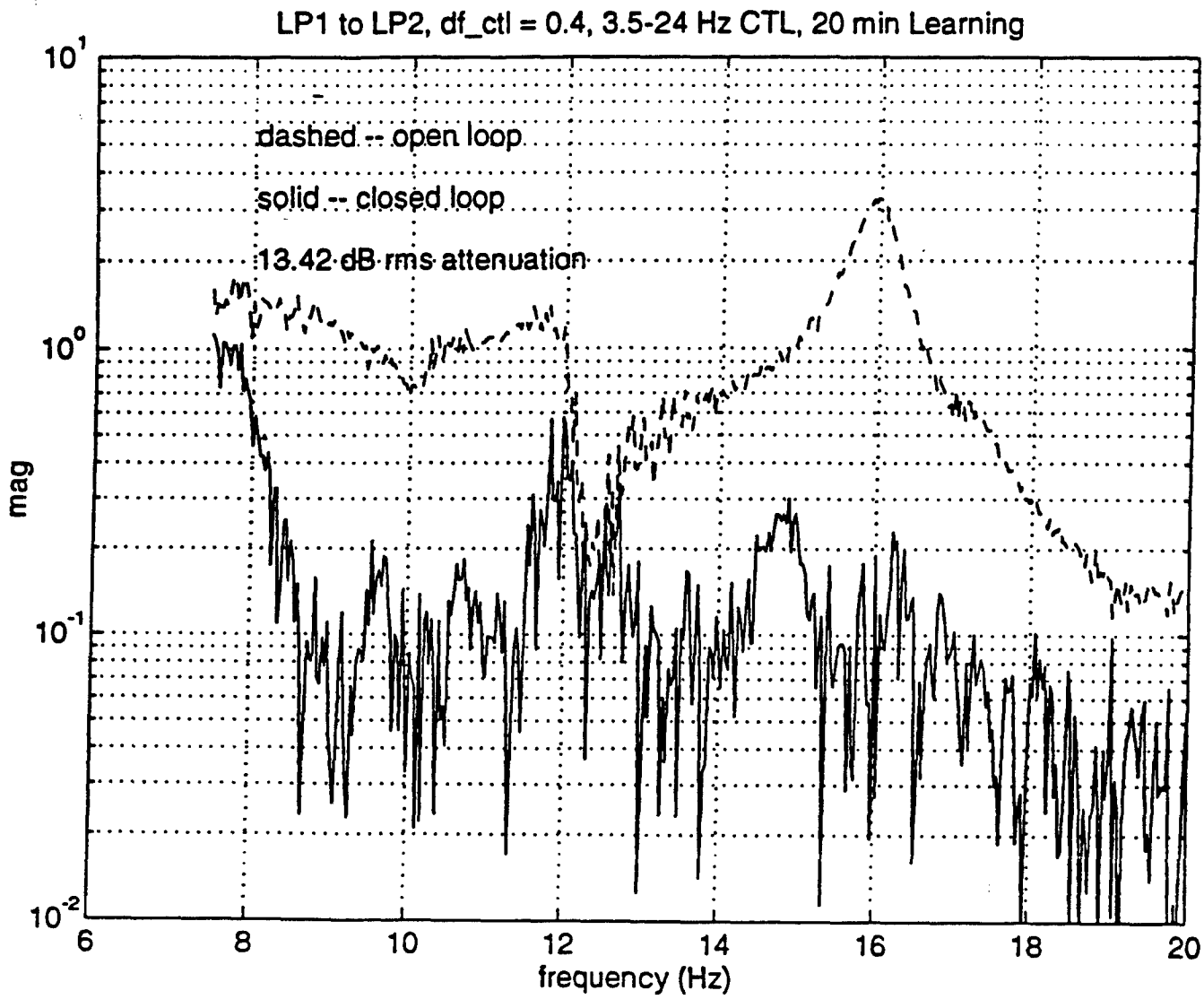


Figure 4.4-3: Same Case as in Figure 4.4-1. Open-Versus Closed-Loop Response After 20 Minutes of Adaptation

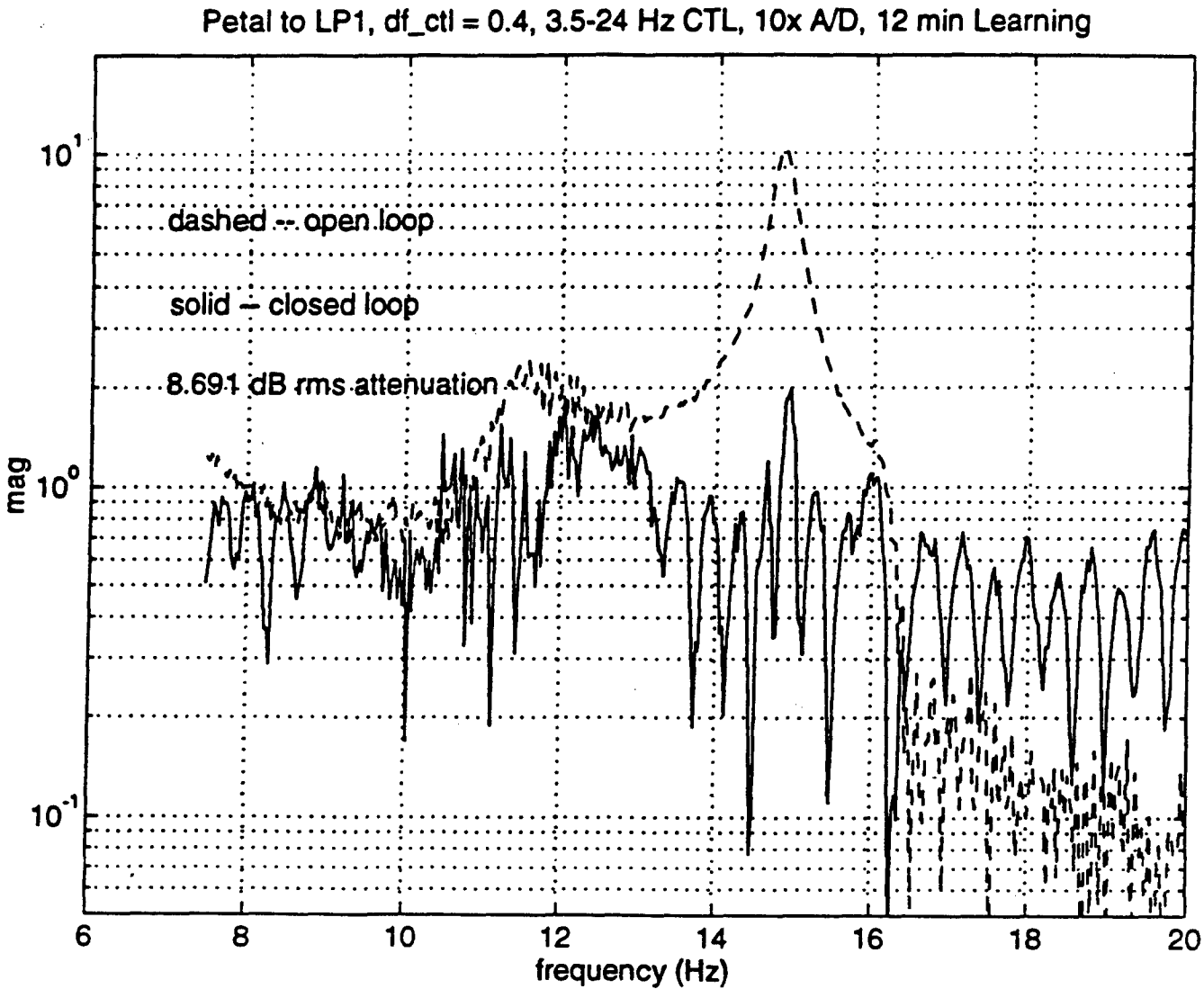


Figure 4.4-4: Petal-Mounted LPACT Disturbance, LPACT 1 Control. Open-Versus Closed-Loop Response After 12 Minutes of Adaptation

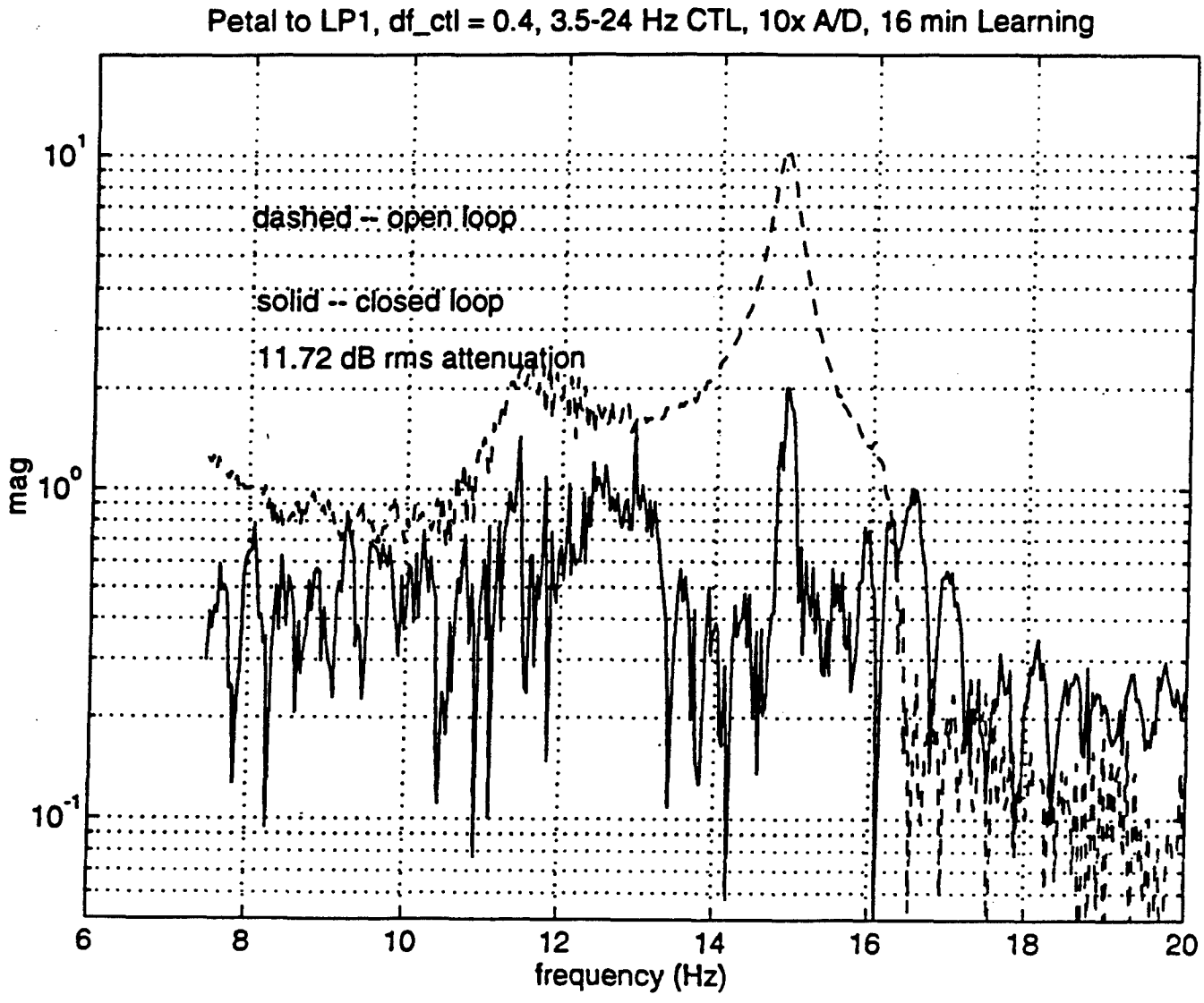


Figure 4.4-5: Same Case as in Figure 4.4-4. Open-Versus Closed-Loop Response After 16 Minutes of Adaptation

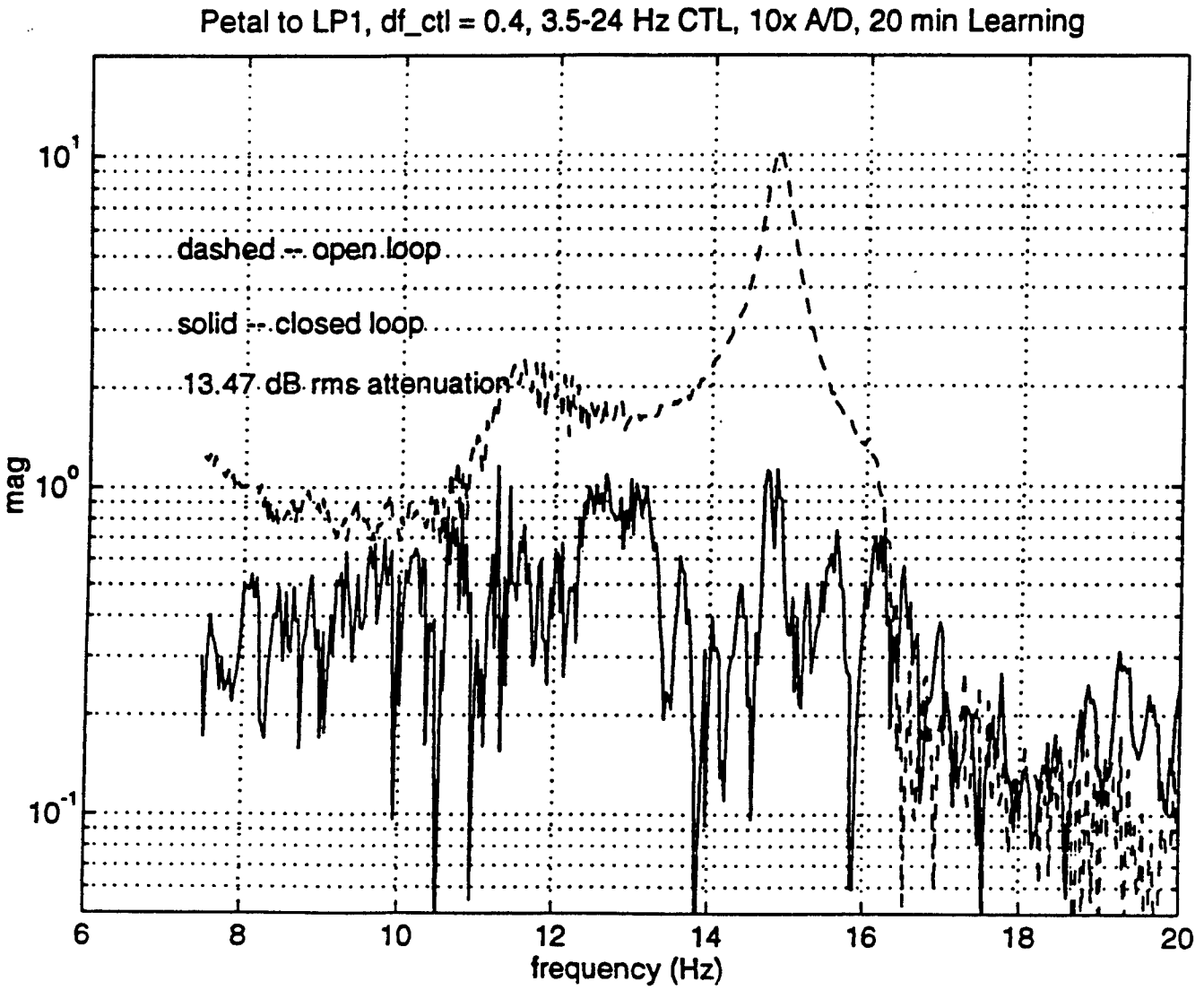


Figure 4.4-6: Same Case as in Figure 4.4-4. Open-Versus Closed-Loop Response After 20 Minutes of Adaptation

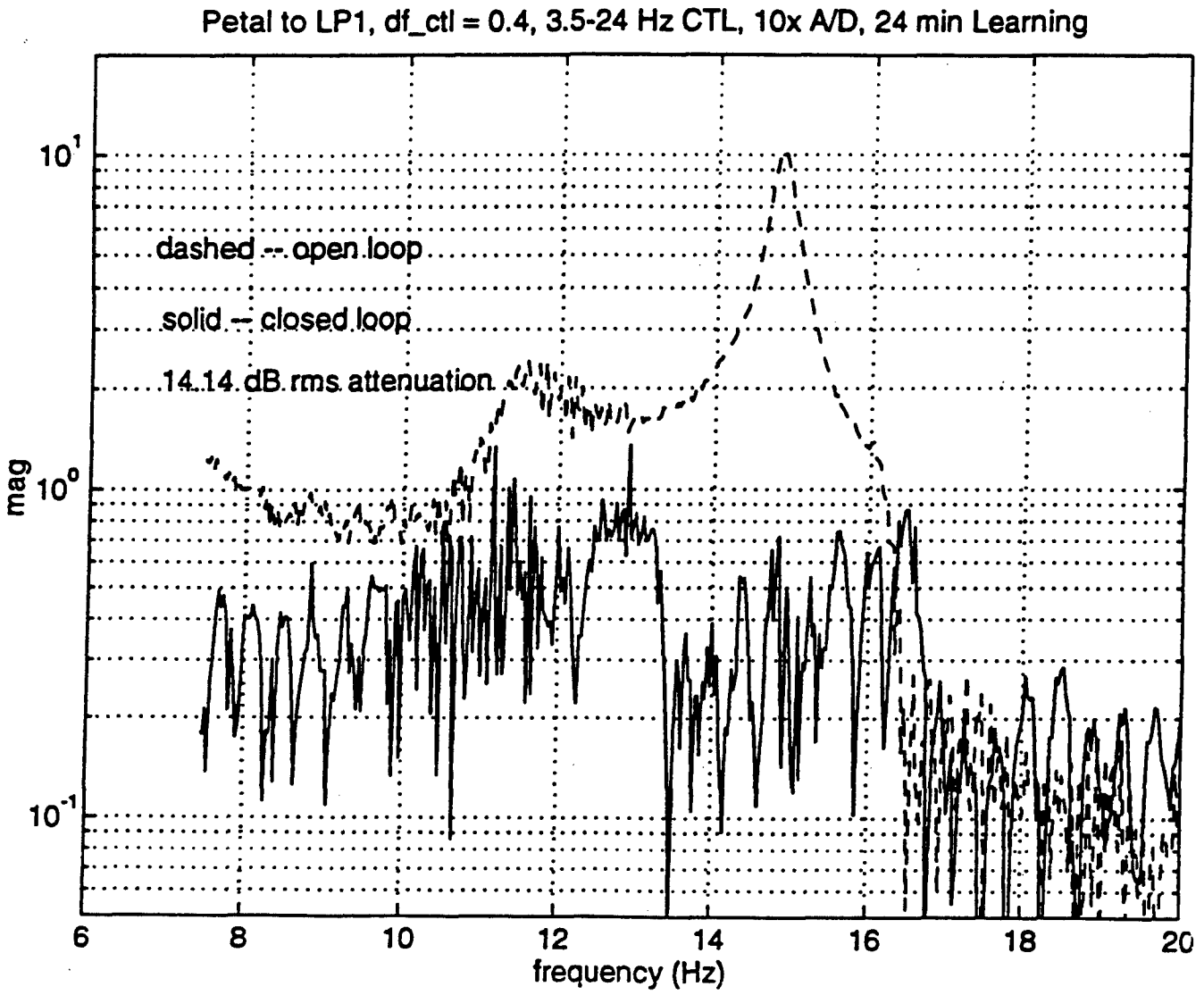


Figure 4.4-7: Same Case as in Figure 4.4-4. Open-Versus Closed-Loop Response After 24 Minutes of Adaptation

5. Amended Option 1 Program Overview

The Amended Option 1 Program addition was dedicated to the development, delivery, and demonstration of an ANC system capable of canceling three tonal vibrations in a fault-tolerant fashion using the three ACESA struts on the ASTREX testbed at the Air Force Phillips Lab. The contract addition was made at the end of June, 1995 and extended the contract to the middle of September, 1996.

This multi-tone demonstration system had two important features that had not been demonstrated previously on the ANC program:

1. Multiple sensors (3) and actuators (6) were used. The sensors were fewer in number than the actuators and were not collocated.
2. In addition to controlling the ASTREX structure without any prior modeling data, the system was designed to be capable of recovering from actuator failures autonomously.

The system that was developed used an architecture that combined analog and digital ANC elements. This resulted in very low throughput required—the sample rate of the digital computer was set at 1/20 Hz. The hardware that was used to implement the algorithm was vastly more capable than required to execute the computations needed by the ANC system. It was chosen to minimize development cost because of commonality with another program.

The system was demonstrated during two test visits, one in January and the other in June of 1996, each for a week.

In the following, we describe the demonstration system in Chapter 6. We then go into detail describing in mathematical terms the algorithms tested in Chapter 7. Chapter 8 describes the experiments that were run and the results achieved. Chapter 9 describes the software that comprised the demonstration system. Chapter 10 gives an overview of the hardware, focusing in some detail on the custom analog cards that were delivered. Finally, Chapter 11 gives a summary of the conclusions and future directions of research for the whole ANC program.

6. ASTREX Demonstration Description

In this chapter we describe the ASTREX demonstration architecture in some detail. We begin with an overview and then proceed to the subsystems that make up the whole.

6.1 Overview of the Demonstration

A summary block diagram of the demonstration is shown in Figure 6-1 below. It consisted of six major components: the ASTREX structure, the ACESA secondary mounting struts and electronics, the Linear Precision Actuators (LPACTS) and servoamps, the three tone disturbance source, the motion sensors and amplifiers, and the ANC control system.

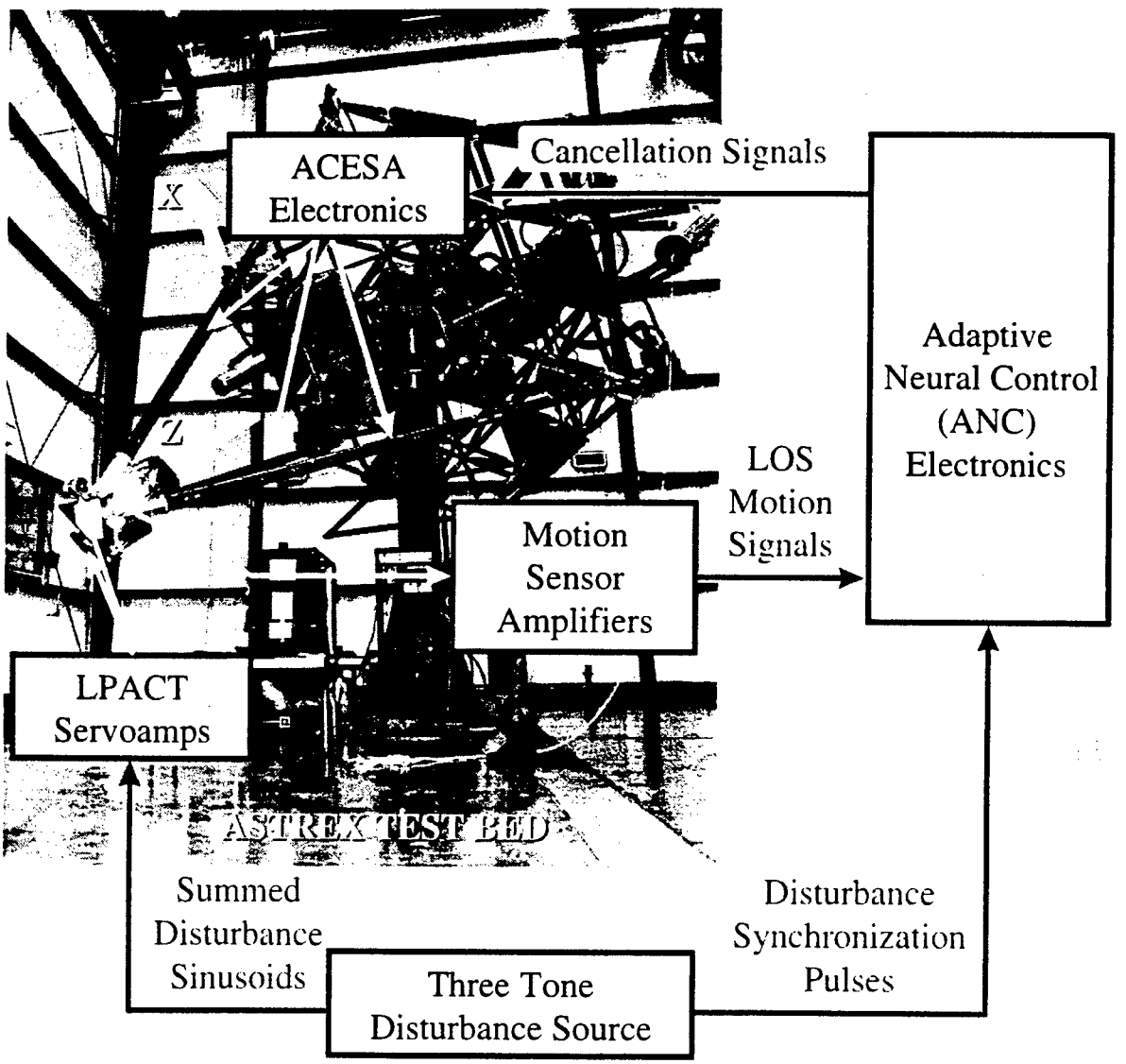


Figure 6-1. Block Diagram of the ASTREX Demonstration

Three sinusoidal disturbances are generated, scaled, and summed in the disturbance source block. These sums are fed to two of the three LPACTs on the secondary tower of the ASTREX test bed as system disturbances (one of the three LPACTs was not functional at the time of test). Meanwhile, three pulse trains, synchronized with each of the three sinusoids, are passed to the ANC controller. Thus, the controller knows the frequency, but not the amplitude or phase of the disturbance inputs to the LPACTs. The controller synthesizes six canceling control signals to feed to the ACESA struts, one for each bending axis of the three struts. The resulting motion of the structure, which is to be canceled out, is sensed and fed to the ANC controller as a training signal.

The demonstration emulates a precision optical system that is disturbed by independent disturbances such as might originate from three control moment gyros (CMGs) with slight imbalances. Motion of key optical subsystems that contribute to line-of-sight (LOS) error are sensed and canceled by the ANC controller using actuators capable of controlling the relevant substructures (in this case, the ACESA struts that support the secondary mirror assembly).

In addition to the components shown in the Figure, a separate system was installed prior to the second test visit to enable visual observation of the cancellation system. This system is depicted in Figure 6-2 below.

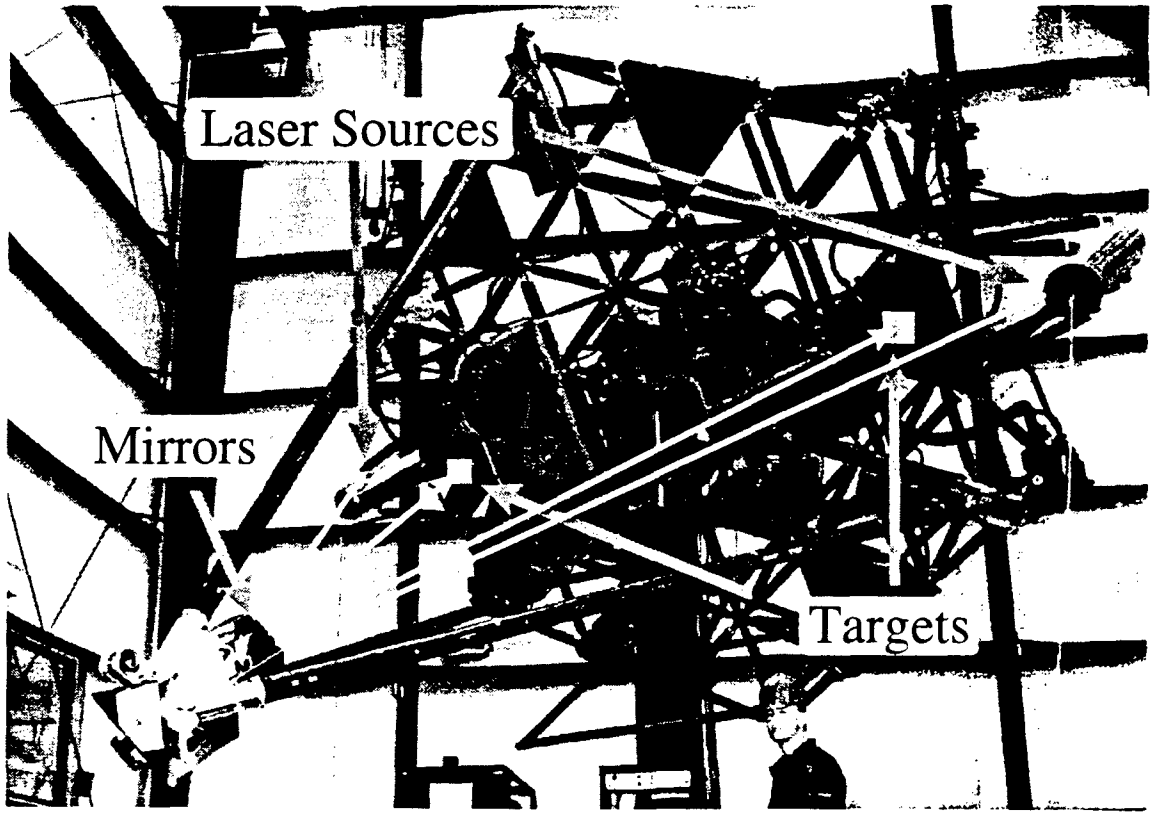


Figure 6-2. Performance Visualization System for the ASTREX Demonstration

Two laser sources are mounted near the cylindrical telescope mass simulators in the figure. These aim at two small mirrors mounted on the secondary assembly. The laser beams are reflected on to paper targets mounted on the primary structure as shown in the figure. This

provides complete monitoring of the effects of the secondary mirror assembly on the LOS performance, and is entirely independent of the sensors used for training the ANC.

We now discuss the various components of the demonstration in somewhat greater detail.

6.2 ACESA Struts and Electronics

The ACESA struts were developed by TRW as part of an earlier effort. The interface to the ANC system is via six analog inputs, which command the three struts, each in two bending axes. The struts are labeled for convenience the 12:00 strut, the 4:00 strut, and the 8:00 strut according to their positions as viewed by an observer looking in the -Z direction (down the boresight from the secondary mirror assembly towards the primary).

Each strut has two inputs: one to cause motion of the secondary in the X direction, and one to cause motion in the Y direction. For safety reasons, the inputs were limited to ±3 V to prevent damage to the structure or struts.

6.3 Motion Sensing for Adaptation

Two different schemes were used during testing to measure the motion of the secondary mirror assembly. The first scheme consisted of two accelerometers mounted in the X and Y directions in the vicinity of LPACT 2. This scheme was sensitive to lateral motion of the secondary and also torsional rotation about the Z axis. It was used during the first test visit and part of the second. Two different sensitivities were used, 10 mV/g and 100 mV/g.

The second scheme consisted of three angular velocity sensors mounted near the mirrors on the secondary. These were insensitive to translation of the tower and provided a much better indication of the effect of the secondary on the visually observed spot motion of the laser beams on the targets. The velocity sensors had sensitivities of approximately 400 mV·sec/rad.

6.4 LPACTS and Disturbance Sources

The disturbance generation system works according to the block diagram shown in Figure 6-3 below.

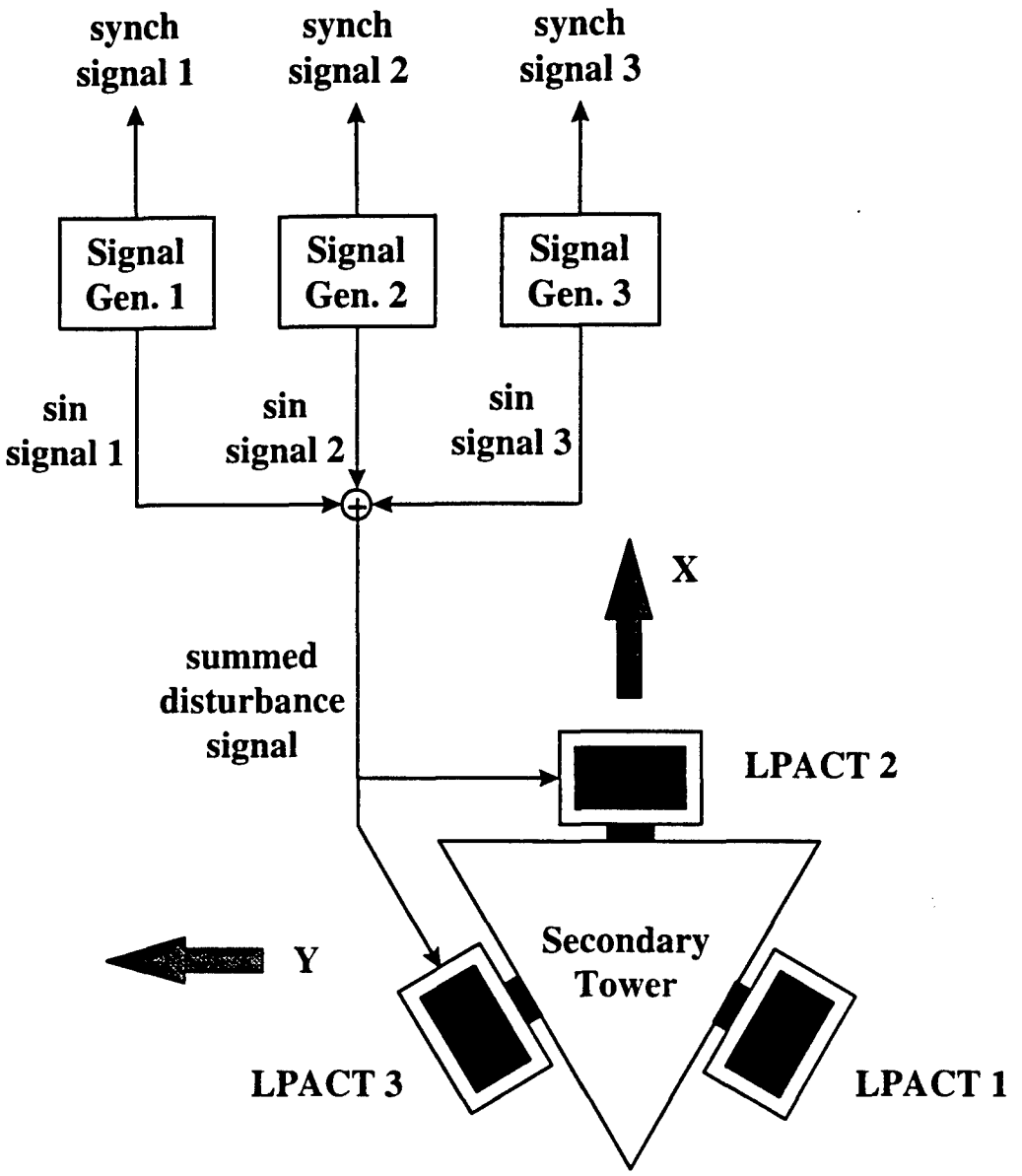


Figure 6-3. The Disturbance System for the ASTREX Demonstration

Three signal generators provide individually adjustable sinusoids, which are summed and input to the two LPACTs used for disturbance generation. The summing function is provided by a simple op-amp circuit. The same disturbance signal is input to both LPACT servo-amps. As mentioned previously, LPACT 1 was inoperative due to an intermittent connection.

The LPACTs provide approximately 10 lb. maximum output for a maximum input of approximately 5 volts at 10 Hz. Slightly more output is available at higher frequencies.

6.5 ANC System

The Adaptive Neural Control System has the structure shown in the block diagram of Figure 6-4 below. Except for the VME computer peripherals (disk units and terminal), the whole assembly fits in a VME card cage.

This implementation of the ANC system is based on the observation that if the control system and the disturbance inputs are the sum of three sinusoids, then the motion sensor outputs must also be the sum of three sinusoids. This is certainly true if the system is linear. Furthermore, the required cancellation signals must be the sum of three sinusoids in this case. Note that any sum of three sinusoids is specified by the six coefficients of the cosines and sines of which it is comprised. The ANC cancellation system is built to determine and operate on these coefficients.

The three disturbance synchronization signals are converted in the largest analog circuit card into sine and cosine signals at each of the three disturbance frequencies. These are used as the fundamental basis functions to process both the motion sensor signals and the control commands.

The motion sensor inputs are amplified and low-pass filtered and then are fed into the adaptive demodulators. The demodulators determine what combination of the three sine and cosine signal pairs are in each of the three motion sensor signals. For each of the three sensors there results six coefficients which are then fed into the A/D converters. It takes approximately 20 seconds for these signals to reach their final values once a new set of demodulated strut commands are issued by the VME computer.

The A/D converter system, residing on one master A/D card and two slave multiplixer cards, converts the 18 coefficients used by the cancellation algorithm to the VME computer for processing. The resolution is 12 bits.

The 12 bit D/A converters, residing on two VME cards, issue 6 commands for each of the six strut bending axes. The 6 commands per axis are the proportions of the 3 pairs of sine or cosine signals which are desired in the strut command for a single strut axis.

The modulator multiplies the 6 coefficients per channel times the appropriate sine and cosine signals to compute the command for each strut's two bending axes.

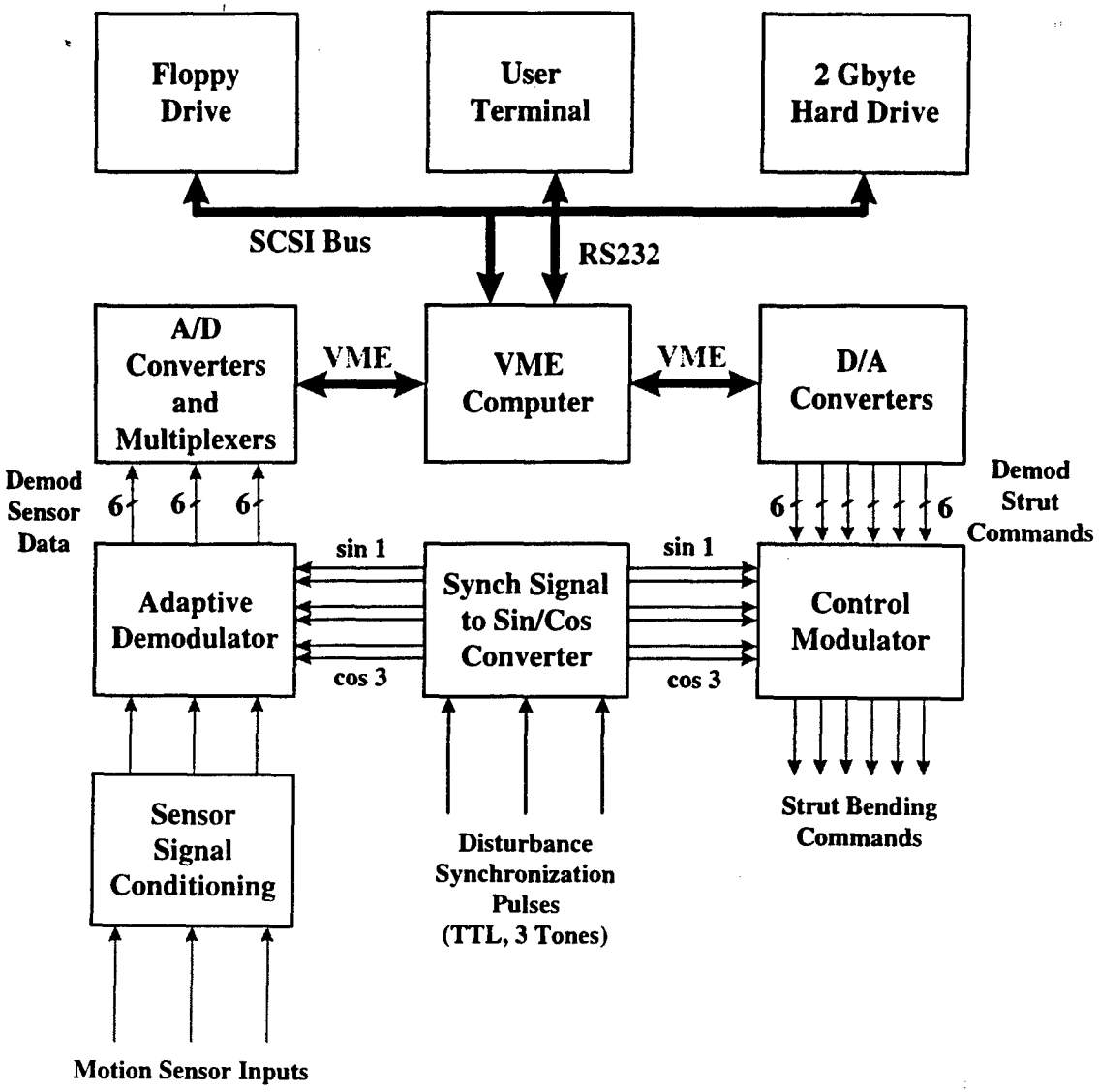


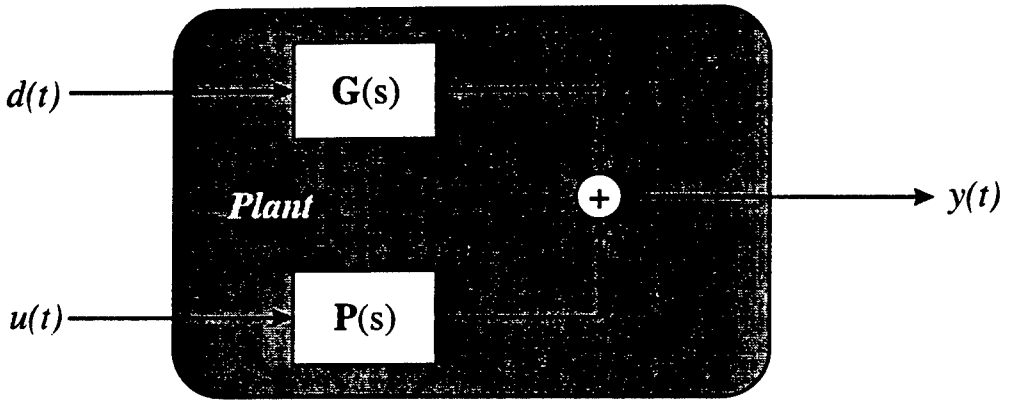
Figure 6-4. Block Diagram of the ANC System

7. ANC Theory and Algorithms

This chapter describes the main theoretical results that form the foundation of the ANC system. Section 7.1 discusses the basic problem setup and notation. The analysis of the analog demodulation system is described in Section 7.2. Section 7.3 covers some important basic results that are used to prove results in the other sections. Finally, Section 7.4 actually describes the algorithms used in the test visits, along with the analysis of convergence in each case.

7.1 Control Problem Formulation and Notation

The basic system setup is shown in Figure 7-1 below.



$$d(t) = \sum_{i=1}^{n} \mathrm{Re}\left\{ d^{(i)} e^{-j\omega_i t} \right\} \qquad (7\text{-}1)$$

$$u(t) = \sum_{i=1}^{n} \mathrm{Re}\left\{ u^{(i)} e^{-j\omega_i t} \right\} \qquad (7\text{-}2)$$

Figure 7-1. The Basic System Notation

The disturbance to the system and the control signal are assumed to be the sum of n sinusoids as shown. The input disturbance, the control vector, and the output are all assumed to be vectors of dimension *nd, nu,* and *ny* respectively. The system is assumed to be linear so that its output is also the sum of sinusoids:

$$y(t) = \sum_{i=1}^{n} \mathrm{Re}\left\{ \mathbf{G}(-j\omega_i) d^{(i)} e^{-j\omega_i t} + \mathbf{P}(-j\omega_i) u^{(i)} e^{-j\omega_i t} \right\} \qquad (7\text{-}3)$$

If we make the following definitions:

$$G^{(i)} \equiv \mathbf{G}(-j\omega_i) d^{(i)} \qquad (7\text{-}4)$$

$$P^{(i)} \equiv \mathbf{P}(-j\omega_i) \tag{7-5}$$

$$y^{(i)} \equiv G^{(i)} + P^{(i)}u^{(i)} \tag{7-6}$$

then we can write

$$y(t) = \sum_{i=1}^{n} \mathrm{Re}\left\{y^{(i)}e^{-j\omega_i t}\right\} \tag{7-7}$$

The goal of the whole exercise must then be to set the expression (7-6) to zero for each of the three tones (i = 1, 2, 3). Note that the problem for each of the three tones is entirely independent.

So far it has been assumed that the coefficients $u^{(i)}$ are constants in time. The approach taken here, however, is to feed the computer estimates of $y^{(i)}$ and to allow it to change $u^{(i)}$ at even intervals sufficiently long in duration to allow the steady state of equation (7-6) to be attained before the onset of a new value. For the ASTREX testbed, a duration of approximately 20 seconds was used in most experiments. Because the control coefficients and the output coefficients measured at the end of the interval are time varying, a new index k must be introduced.

$$y^{(i)}(k) = G^{(i)} + P^{(i)}u^{(i)}(k) \tag{7-8}$$

Also, since the problem of controlling several tones is entirely uncoupled, we drop the superscript (i) when considering the problem of (7-8) for any one tone.

7.2 Analog Demodulation System and Analysis

We have already mentioned that the computer does not get access to y(t) directly, but only to estimates of the $y^{(i)}(k)$. We now describe how these estimates are obtained in analog electronics. The assumption is made that we are in the steady state so that equation (7-7) holds. Naturally, we must wait long enough for this to be a good approximation.

Define:

$$q \equiv \left[\mathrm{Re}\left\{y^{(1)}\right\} \;\; \mathrm{Im}\left\{y^{(1)}\right\} \;\; \cdots \;\; \mathrm{Re}\left\{y^{(n)}\right\} \;\; \mathrm{Im}\left\{y^{(n)}\right\}\right]^T \tag{7-9}$$

$$v(t) \equiv \left[\cos(\omega_1 t) \;\; \sin(\omega_1 t) \;\; \cdots \;\; \cos(\omega_n t) \;\; \sin(\omega_n t)\right]^T \tag{7-10}$$

It is then just a matter of expanding (7-7) for a single scalar element to see that

$$y(t) = v(t)^T q \tag{7-11}$$

Now construct an estimate of q as follows:

$$e(t) = y(t) - v(t)^T \hat{q}(t) \tag{7-12}$$

$$\dot{\hat{q}}(t) = \lambda\, v(t)e(t) \tag{7-13}$$

We can find out what happens to the error:

$$\begin{aligned}
&\dot{\hat{q}}(t) = \lambda\, v(t)e(t) \\
&\Delta q(t) \equiv \hat{q}(t) - q \\
&\Delta\dot{q}(t) = -\lambda\, v(t)v(t)^T \Delta q(t) \\
&\frac{d}{dt}\left\|\Delta q(t)\right\|^2 = -\lambda \left(v(t)^T \Delta q(t)\right)^2 \le 0
\end{aligned} \tag{7-14}$$

This implies that the error magnitude is monotonically nonincreasing. In fact, a little more analysis shows that it converges precisely to zero. Before proceeding, however, it is useful to show a block diagram of the demodulation system, because it is essential to a later understanding of the analog electronics used to implement it:

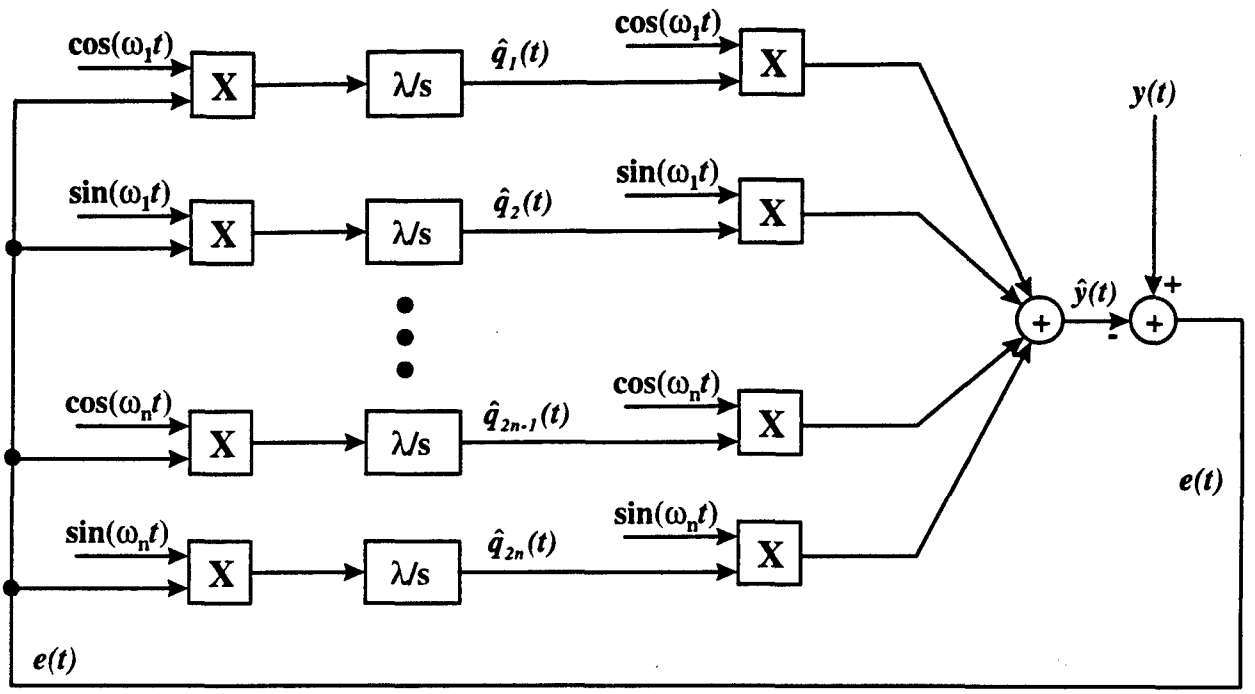


Figure 7-2. Block Diagram of the Multi-Tone Demodulator

Clearly, analog multipliers form an important part of this device. We now commence an analysis of this network from a different point of view that provides detailed information on the convergence behavior and sensitivity to errors. This analysis is done in the Fourier Transform domain with the following conventions on transforms:

$$Y(\omega) = \int_{-\infty}^{\infty} y(t)e^{-j\omega t}dt, \quad y(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} Y(\omega)e^{j\omega t}d\omega \tag{7-15}$$

It will be useful to use complex notation in what follows. To that end define:

$$\hat{y}^{(i)}(t) \equiv \hat{q}_{2i-1}(t) + j\,\hat{q}_{2i}(t) \tag{7-16}$$

Keeping close track of complex operations, the estimator equations are then:

$$\hat{y}(t) = \sum_{i=1}^{n} \mathrm{Re}\left\{\hat{y}^{(i)}(t) e^{-j\omega_i t}\right\} \tag{7-17}$$

$$e(t) = y(t) - \hat{y}(t) \tag{7-18}$$

$$\dot{\hat{y}}^{(i)}(t) = \lambda\, e^{j\omega_i t} e(t) + \lambda w_i \tag{7-19}$$

where we have included an extra offset in the integrator of (7-19) because it is an important source of error in the electronics. We will make frequent use of the Dirac delta function in the following. There are several important identities required for our calculations:

$$x(t) = e^{j\omega_i t} \quad \Leftrightarrow \quad X(\omega) = 2\pi\delta(\omega - \omega_i) \tag{7-20a}$$

$$x(t) = \mathrm{Re}\{y(t)\} \quad \Leftrightarrow \quad X(\omega) = \tfrac{1}{2}\left(Y(\omega) + Y(-\omega)^*\right) \tag{7-20b}$$

$$x(t) = y(t) e^{j\omega_i t} \quad \Leftrightarrow \quad X(\omega) = Y(\omega - \omega_i) \tag{7-20c}$$

The idea is to solve for the Fourier Transforms of the coefficient estimates as a function of the true coefficients and the integrator offsets. To that end, we follow the signals around the loop, beginning with the input *y*(t).

Equations (7-7) and (7-20a), (7-20b) imply:

$$Y(\omega) = \pi \sum_{i=1}^{n} \left(y^{(i)} \delta(\omega + \omega_i) + y^{(i)*} \delta(\omega - \omega_i) \right) \tag{7-21}$$

Equations (19) and (20c) imply:

$$\hat{Y}^{(i)}(\omega) = \frac{2\lambda\pi w_i}{j\omega} \delta(\omega) + \frac{\lambda}{j\omega} E(\omega - \omega_i) \tag{7-22}$$

noting that, because *e*(t) is real, $E(\omega) = E(-\omega)^*$. Equations (7-17) and (7-20b,c) imply:

$$\hat{Y}(\omega) = \tfrac{1}{2} \sum_{i=1}^{n} \left(\hat{Y}^{(i)}(\omega + \omega_i) + \hat{Y}^{(i)}(-\omega + \omega_i)^* \right) \tag{7-23}$$

Now we plug (7-22) into (7-23) and clean up the mess:

$$\hat{Y}(\omega) = \lambda\pi \sum_{i=1}^{n} \left(w_i \frac{\delta(\omega + \omega_i)}{j(\omega + \omega_i)} + w_i^* \frac{\delta(\omega - \omega_i)}{j(\omega - \omega_i)} \right) + \left[\sum_{i=1}^{n} \frac{j\lambda\omega}{\omega_i^2 - \omega^2} \right] E(\omega) \tag{7-24}$$

Now use (7-18) to get:

$$\hat{Y}(\omega) = Y(\omega) - E(\omega) =$$

$$\lambda\pi \sum_{i=1}^{n} \left(w_i \frac{\delta(\omega + \omega_i)}{j(\omega + \omega_i)} + w_i^* \frac{\delta(\omega - \omega_i)}{j(\omega - \omega_i)} \right) + \left[\sum_{i=1}^{n} \frac{j\lambda\omega}{\omega_i^2 - \omega^2} \right] E(\omega) \tag{7-24}$$

$$\left[1 + \sum_{i=1}^{n} \frac{j\lambda\omega}{\omega_i^2 - \omega^2} \right] E(\omega) = Y(\omega) - \lambda\pi \sum_{i=1}^{n} \left(w_i \frac{\delta(\omega + \omega_i)}{j(\omega + \omega_i)} + w_i^* \frac{\delta(\omega - \omega_i)}{j(\omega - \omega_i)} \right)$$

To get further define:

$$H(\omega) \equiv \left(1 + \sum_{i=1}^{n} \frac{j\lambda\,\omega}{\omega_i^{\,2} - \omega^2}\right)^{-1} \tag{7-25}$$

Note this is zero at the tonal frequencies $\pm\omega_i$. Consider the contribution of

$$\begin{aligned}
&\left[1 + \sum_{i=1}^{n} \frac{j\lambda\,\omega}{\omega_i^{\,2} - \omega^2}\right]^{-1} w_i \frac{\delta\,(\omega + \omega_i)}{j(\omega + \omega_i)} \\
&= \frac{j(\omega + \omega_i)\,j(-\omega_i - \omega_i)}{-j\lambda\,\omega_i} w_i \frac{\delta\,(\omega + \omega_i)}{j(\omega + \omega_i)} \\
&= 2\frac{w_i}{\lambda}\delta\,(\omega + \omega_i)
\end{aligned} \tag{7-26}$$

Since only the values at the zero argument of the delta function are relevant. Similarly,

$$\begin{aligned}
&\left[1 + \sum_{i=1}^{n} \frac{j\lambda\,\omega}{\omega_i^{\,2} - \omega^2}\right]^{-1} w_i^{*} \frac{\delta\,(\omega - \omega_i)}{j(\omega - \omega_i)} \\
&= \frac{j(\omega_i + \omega_i)\,j(\omega - \omega_i)}{j\lambda\,\omega_i} w_i^{*} \frac{\delta\,(\omega - \omega_i)}{j(\omega - \omega_i)} \\
&= 2\frac{w_i^{*}}{\lambda}\delta\,(\omega - \omega_i)
\end{aligned} \tag{7-27}$$

To summarize,

$$E(\omega) = H(\omega)Y(\omega) - 2\pi\sum_{i=1}^{n}\left(w_i\delta\,(\omega + \omega_i) + w_i^{*}\delta\,(\omega - \omega_i)\right) \tag{7-28}$$

Plugging this result into (7-22) we get:

$$\begin{aligned}
\hat{Y}^{(i)}(\omega) &= \frac{2\pi\lambda\,w_i}{j\omega}\delta\,(\omega) + \frac{\lambda}{j\omega}H(\omega - \omega_i)Y(\omega - \omega_i) \\
&- 2\pi\frac{\lambda}{j\omega}\sum_{p=1}^{n}\left(w_p\delta\,(\omega - \omega_i + \omega_p) + w_p^{*}\delta\,(\omega - \omega_i - \omega_p)\right) \\
&= \frac{\lambda}{j\omega}H(\omega - \omega_i)Y(\omega - \omega_i) + \lambda\,\pi\frac{w_i^{*}}{j\omega_i}\delta\,(\omega - 2\omega_i) \\
&- \frac{2\lambda\,\pi}{j}\sum_{p\neq i}\left(\frac{w_p}{\omega_i - \omega_p}\delta\,(\omega - (\omega_i - \omega_p)) + \frac{w_p^{*}}{\omega_i + \omega_p}\delta\,(\omega - (\omega_i + \omega_p))\right)
\end{aligned} \tag{7-29}$$

Some final work is needed to compute what goes on due to the *Y* input. We know that *H* is zero whenever its argument is +/- an input frequency. Because *Y* is the sum only of such sinusoids, we need examine only those frequencies. The only nonzero case is for zero frequency, at which we have:

$$\frac{\lambda}{j\omega}\hat{H}(\omega-\omega_i)Y(\omega-\omega_i)=\frac{\lambda}{j\omega}\left(\frac{2j\omega}{\lambda}\right)\left(\pi\, y^{(i)}\delta(\omega)\right)$$
$$=2\pi\delta(\omega)\,y^{(i)} \tag{7-30}$$

which means the coefficient is identified perfectly except for the stuff in (7-28). The final expression for the coefficient is:

$$\hat{Y}^{(i)}(\omega)=2\pi\delta(\omega)y^{(i)}+\lambda\pi\frac{w_i^*}{j\omega_i}\delta(\omega-2\omega_i)$$
$$-\frac{2\lambda\pi}{j}\sum_{p\neq i}\left(\frac{w_p}{\omega_i-\omega_p}\delta\left(\omega-(\omega_i-\omega_p)\right)+\frac{w_p^*}{\omega_i+\omega_p}\delta\left(\omega-(\omega_i+\omega_p)\right)\right) \tag{7-31}$$

The DC value of the coefficient is exactly right, but integrator offsets produce oscillatory behavior at the sum and difference frequencies of the tones, with magnitude proportional to the integrator gain and inversely proportional to the frequency difference.

7.3 Helpful Theory for Cancellation Algorithms

This section gives some essential results that are used to prove convergence for the algorithms that were tested on ASTREX.

7.3.1 Standard ANC Learning Law Update Results

In this section we consider results for the simple gradient descent algorithm with appropriate scaling. These results are used to show convergence for the later, more complex algorithms.

We begin by describing the plant:

$$z(k)=C(k)x, \tag{7-32}$$

where $z(k)$ is a complex vector measurement sequence, x is a complex parameter vector to be estimated, and $C(k)$ is a complex matrix. We define the following estimator:

$$\hat{x}(k):\quad \hat{x}(k+1)=\hat{x}(k)+s(k)^2C(k)^H(z(k)-C(k)\hat{x}(k)),\ \hat{x}(0)=0,$$
$$s(k)\text{ is a real scalar such that }\ |s(k)|\|C(k)\|\leq 1. \tag{7-33}$$

In addition, define the following for convenience:

$$\Delta x(k)\equiv\hat{x}(k)-x$$
$$\hat{z}(k)\equiv C(k)\hat{x}(k) \tag{7-34}$$
$$e(k)\equiv z(k)-\hat{z}(k)=-C(k)\Delta x(k)$$

With these definitions, the update equation (7-33) can be simplified, and we can write expressions for the parameter error update:

$$\hat{x}(k+1)=\hat{x}(k)+s(k)^2C(k)^He(k)=\hat{x}(k)-s(k)^2C(k)^HC(k)\Delta x(k)$$
$$\Delta x(k+1)=\Delta x(k)+s(k)^2C(k)^He(k)=(I-s(k)^2C(k)^HC(k))\Delta x(k) \tag{7-35}$$

Before stating the main result, we start with a definition and some simple consequences.

Definition 1

A sequence of complex n-vectors $x(k)$ is L_2 iff there exists a limit S and an $M < \infty$ for every $\delta > 0$ such that $0 \le S - \sum_{k=0}^{r} \|x(k)\|^2 < \delta$ for all $r \ge M$. In other words, $\sum_{k=0}^{\infty} \|x(k)\|^2 = S < \infty$.

Proposition 1

The elements of an L_2 sequence converge to zero, and the sum of an L_2 sequence converges to a finite limit.

Proof:

By "converge to zero" we mean we can always choose an index in the sequence M such that all elements of the sequence after it are smaller than any value δ we care to name.

Using the same choice of M as in the definition,

$S - \sum_{k=0}^{r} \|x(k)\|^2 < \delta$ and $0 \le S - \sum_{k=0}^{r+1} \|x(k)\|^2 \Rightarrow \|x(r+1)\|^2 \le S - \sum_{k=0}^{r} \|x(k)\|^2 < \delta$, true for any $r \ge M$.

Also,

$$\left\| \sum_{k=0}^{\infty} x(k) \right\|^2 \le 2 \sum_{k=0}^{\infty} \|x(k)\|^2 = S < \infty .$$

Proposition 2

If, for a sequence of complex n-vectors $X \equiv \{x(0), x(1), \cdots\}$, $\sum_{k=0}^{r} \|x(k)\|^2 < a$ uniformly for all r, then X is an L_2 sequence.

Proof:

The sequence of sums $S_r \equiv \sum_{k=0}^{r} \|x(k)\|^2$, $r = 0,1,\cdots$ is monotonically nondecreasing and bounded from above by a. Hence it converges to a bounded limit $\le a$.

We now proceed with the main result relating to the estimator of equation (33). These are deterministic results independent of conditions on the matrices $C(k)$.

Proposition 3

The parameter error $\|\Delta x(k)\|$ is monotonically nonincreasing, $s(k)e(k)$ is an L_2 sequence convergent to zero, and both $\hat{x}(k)$ and $\Delta x(k)$ converge to definite limits.

Proof:

Use the scaling factor $s(k)$ and the expression (7-35b) to obtain a bound on the new parameter error:

$$\begin{aligned}
\|\Delta x(k+1)\|^2 &= \Delta x(k)^H (I - s(k)^2 C(k)^H C(k))(I - s(k)^2 C(k)^H C(k))\Delta x(k) \\
&= \Delta x(k)^H (I - 2s(k)^2 C(k)^H C(k) + s(k)^4 C(k)^H C(k) C(k)^H C(k))\Delta x(k) \\
&= \|\Delta x(k)\|^2 - 2\|s(k)C(k)\Delta x(k)\|^2 + \|s(k)^2 C(k)^H C(k)\Delta x(k)\|^2 \\
&\le \|\Delta x(k)\|^2 - 2\|s(k)C(k)\Delta x(k)\|^2 + \|s(k)C(k)^H\|^2 \|s(k)C(k)\Delta x(k)\|^2 \\
&\le \|\Delta x(k)\|^2 - 2\|s(k)C(k)\Delta x(k)\|^2 + \|s(k)C(k)\Delta x(k)\|^2 \\
&= \|\Delta x(k)\|^2 - \|s(k)C(k)\Delta x(k)\|^2
\end{aligned}$$

Or, summarizing:

$$\|\Delta x(k+1)\|^2 \le \|\Delta x(k)\|^2 - \|s(k)C(k)\Delta x(k)\|^2 = \|\Delta x(k)\|^2 - \|s(k)e(k)\|^2 \tag{7-36}$$

which cannot be an increase. The first part is proved. By iterating (7-36) we get:

$$\begin{aligned}
&\|\Delta x(k+1)\|^2 \le \|\Delta x(k)\|^2 - \|s(k)e(k)\|^2 \\
&\|\Delta x(k+1)\|^2 \le \|\Delta x(k-1)\|^2 - \|s(k-1)e(k-1)\|^2 - \|s(k)e(k)\|^2 \qquad (7\text{-}37) \\
&\|\Delta x(k+1)\|^2 \le \|\Delta x(0)\|^2 - \sum_{m=0}^{k} \|s(m)e(m)\|^2 \ge 0 \quad \forall k > 0
\end{aligned}$$

Placing the summation on the right of the last inequality establishes the second part of the proposition, that $s(k)e(k)$ is an L_2 sequence and must converge to zero.

Now use (7-35a) for the parameter estimate updates to write:

$$\begin{aligned}
&\hat{x}(k) = \sum_{i=0}^{k-1} s(i)^2 C(i)^H e(i) = \sum_{i=0}^{k-1} \big(s(i)C(i)^H\big)\big(s(i)e(i)\big), \\
&\big\|\big(s(i)C(i)^H\big)\big(s(i)e(i)\big)\big\| \le \|s(i)e(i)\|
\end{aligned} \tag{7-38}$$

By the comparison test, the parameter estimate sequence $\hat{x}(k)$ is the sum of an L_2 sequence, so it must achieve a definite, bounded limit. So then also must the parameter error vector $\Delta x(k)$, since it differs from the parameter estimate only by a constant.

Notice that these results do not guarantee that the weight estimates actually converge to their correct values, but only that the norm of the error decreases and that the weights converge to some definite value. To ensure convergence to the correct weights requires that the input *C(k)* be "sufficiently rich". The following gives one example.

Proposition 4

Suppose $s(k)C(k)$ converges to a sequence of matrices $\Gamma(k)$ periodic with period q such that

$$rank\left(\begin{bmatrix}\Gamma(k+1)\\ \vdots \\ \Gamma(k+q)\end{bmatrix}\right) = nx$$

Then $\hat{x}(k) \to x, \quad \Delta x(k) \to 0$.

Proof:

Study the L_2 sequence:

$$s(k)e(k) = -s(k)C(k)\Delta x(k) \to -\Gamma(k)\Delta x_\infty \to 0.$$

The assumption, however, guarantees that there can be no way of escaping $\Delta x_\infty = 0$.

Also, the following nonlinear equation trick can be useful in adjusting a learning stimulus.

Proposition 5

The equation:

$$x(k) = \frac{bx(k-1)}{\left(b^\alpha + x(k-1)^\alpha\right)^{1/\alpha}}, \quad k>0, \quad x(0)>0, \quad b>0, \quad \alpha \neq 0 \tag{7-39}$$

has the solution:

$$x(k) = \frac{bx(0)}{\left(b^\alpha + kx(0)^\alpha\right)^{1/\alpha}}, \quad k \geq 0. \tag{7-40}$$

If $\alpha \geq 2$, then

$$\lim_{N\to\infty}\sum_{k=0}^{N}|x(k)|^2 = \infty$$

and this is <u>not</u> an L_2 sequence, though it converges to zero.

Proof:

First,

$$x(0) = \frac{bx(0)}{\left(b^\alpha + 0x(0)^\alpha\right)^{1/\alpha}} = \frac{bx(0)}{\left(b^\alpha\right)^{1/\alpha}} = x(0).$$

Then,

$$x(k)=\frac{bx(k-1)}{\left(b^{\alpha}+x(k-1)^{\alpha}\right)^{1/\alpha}}=\frac{b\dfrac{bx(0)}{\left(b^{\alpha}+(k-1)x(0)^{\alpha}\right)^{1/\alpha}}}{\left(b^{\alpha}+\left(\dfrac{bx(0)}{\left(b^{\alpha}+(k-1)x(0)^{\alpha}\right)^{1/\alpha}}\right)^{\alpha}\right)^{1/\alpha}}$$

$$x(k)^{\alpha}=\frac{b^{\alpha}\dfrac{b^{\alpha}x(0)^{\alpha}}{\left(b^{\alpha}+(k-1)x(0)^{\alpha}\right)}}{b^{\alpha}+\dfrac{b^{\alpha}x(0)^{\alpha}}{\left(b^{\alpha}+(k-1)x(0)^{\alpha}\right)}}$$

$$=\frac{b^{\alpha}b^{\alpha}x(0)^{\alpha}}{b^{\alpha}\left(b^{\alpha}+(k-1)x(0)^{\alpha}\right)+b^{\alpha}x(0)^{\alpha}}=\frac{b^{\alpha}x(0)^{\alpha}}{b^{\alpha}+kx(0)^{\alpha}}$$

$$x(k)=\frac{bx(0)}{\left(b^{\alpha}+kx(0)^{\alpha}\right)^{1/\alpha}}=\frac{b}{\left(\dfrac{b^{\alpha}}{x(0)^{\alpha}}+k\right)^{1/\alpha}}$$

and the proof by induction is finished for the formulae (7-39) and (7-40). Now, as to the fact that the sequence is not L_2 for $\alpha \geq 2$ consider:

$$\sum_{k=0}^{\infty}\left|\frac{bx(0)}{\left(b^{\alpha}+kx(0)^{\alpha}\right)^{1/\alpha}}\right|^{2}=\sum_{k=0}^{\infty}\frac{\left(bx(0)\right)^{2}}{\left(b^{\alpha}+kx(0)^{\alpha}\right)^{2/\alpha}}=\sum_{k=0}^{\infty}\int_{k}^{k+1}\frac{\left(bx(0)\right)^{2}}{\left(b^{\alpha}+kx(0)^{\alpha}\right)^{2/\alpha}}dr$$

$$\geq\sum_{k=0}^{\infty}\int_{k}^{k+1}\frac{\left(bx(0)\right)^{2}}{\left(b^{\alpha}+rx(0)^{\alpha}\right)^{2/\alpha}}dr=\int_{0}^{\infty}\frac{\left(bx(0)\right)^{2}}{\left(b^{\alpha}+rx(0)^{\alpha}\right)^{2/\alpha}}dr=\int_{0}^{\infty}\frac{\left(bx(0)\right)^{2}}{\left(b^{\alpha}+rx(0)^{\alpha}\right)^{2/\alpha}}dr$$

$$=\frac{\left(bx(0)\right)^{2}}{x(0)^{\alpha}}\int_{b^{\alpha}}^{\infty}\frac{1}{s^{2/\alpha}}ds=\frac{\left(bx(0)\right)^{2}}{x(0)^{\alpha}}\begin{cases}\log(s)\Big|_{b^{\alpha}}^{\infty} & \alpha=2\\ \dfrac{\alpha}{\alpha-2}s^{\frac{\alpha-2}{\alpha}}\Big|_{b^{\alpha}}^{\infty} & \text{else}\end{cases}=\infty\quad\alpha\geq 2$$

7.3.2 Kalman Filtering and Convergence Results

The foundation of the Kalman Filter is in probability theory. It can be derived in numerous ways. Here we restate results based on Gaussian assumptions (see [8] for the proof). We use the brackets $\langle\ \rangle$ to denote expectation over all random variables.

Proposition 6

Suppose $x(k)$ is an nx -dimensional, complex Gaussian stochastic process with initial statistics $\langle x(0)\rangle = \bar{x}_0$ and $Q_0 = \left\langle (x(0)-\bar{x}_0)(x(0)-\bar{x}_0)^H \right\rangle > 0$. Suppose $w(k)$ and $v(k)$ are white, Gaussian, zero-mean processes of dimension nx and ny, and with covariances $W(k)$ and $V(k)$, respectively, independent of each other and x. Suppose $y(k)$ is an ny-dimensional, complex sequence of measurements, and that $A(k)$ and $C(k)$ are known complex sequences of matrices of appropriate dimensions to fit into the plant:

$$\begin{aligned} x(k+1) &= A(k)x(k)+w(k) \\ y(k) &= C(k)x(k)+v(k) \end{aligned} \tag{7-41}$$

Then the ***least mean square error estimator*** using the measurements is constructed as a two-step process, a measurement update followed by a prediction. Initialization is begun with:

$$\begin{aligned} \tilde{x}(0) &= x_0 \\ \tilde{Q}(0) &= Q_0 \end{aligned} \tag{7-42}$$

Measurement updates proceed with:

$$\hat{Q}(k) = \tilde{Q}(k) - \tilde{Q}(k)C(k)^H\left(C(k)\tilde{Q}(k)C(k)^H + V(k)\right)^{-1}C(k)\tilde{Q}(k) \tag{7-43a}$$

$$\hat{x}(k) = \tilde{x}(k) + \tilde{Q}(k)C(k)^H\left(C(k)\tilde{Q}(k)C(k)^H + V(k)\right)^{-1}\left(y(k) - C(k)\tilde{x}(k)\right) \tag{7-43b}$$

$$k \geq 0$$

Predictions proceed with:

$$\begin{aligned} \tilde{Q}(k+1) &= A(k)\hat{Q}(k)A(k)^H + W(k) \\ \tilde{x}(k+1) &= A(k)\hat{x}(k) \end{aligned} \tag{7-44}$$

$$k \geq 0$$

The variables $\hat{x}(k), \hat{Q}(k)$ are the expected value of the state and its covariance given the measurement, and $\tilde{x}(k), \tilde{Q}(k)$ are these just prior to measurement, respectively.

There are a few simplifying expressions that can be used in the update equations that are useful. These are shown in Proposition 7.

Proposition 7

Suppose $V(k)$ is invertible. Then equations (7-43) can be rewritten:

$$\begin{aligned} \hat{Q}(k) &= \left(\tilde{Q}(k)^{-1} + C(k)^H V(k)^{-1} C(k)\right)^{-1} \\ \hat{x}(k) &= \tilde{x}(k) + \hat{Q}(k)C(k)^H V(k)^{-1}\left(y(k) - C(k)\tilde{x}(k)\right) \end{aligned} \tag{7-45}$$

Proof:

$$\tilde{Q}(k)-\tilde{Q}(k)C(k)^H\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)^{-1}C(k)\tilde{Q}(k)=\left(\tilde{Q}(k)^{-1}+C(k)^HV(k)^{-1}C(k)\right)^{-1}$$

because of the infamous Matrix Inversion Lemma.

To resolve (7-43b) multiply (7-43a) on the right by $C(k)^HV(k)^{-1}$ to get a new expression for the multiplier of the measurement error:

$$\left[\tilde{Q}(k)-\tilde{Q}(k)C(k)^H\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)^{-1}C(k)\tilde{Q}(k)\right]C(k)^HV(k)^{-1}$$

$$=\tilde{Q}(k)C(k)^H\left[I-\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)^{-1}C(k)\tilde{Q}(k)C(k)^H\right]V(k)^{-1}$$

$$=\tilde{Q}(k)C(k)^H\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)^{-1}$$

$$\times\left[\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)-C(k)\tilde{Q}(k)C(k)^H\right]V(k)^{-1}$$

$$=\tilde{Q}(k)C(k)^H\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)^{-1}\left(V(k)\right)V(k)^{-1}$$

$$=\tilde{Q}(k)C(k)^H\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)^{-1}$$

This completes the proof.

Next, we write some interesting results on statistics that estimate *V(k)*. These are useful for on-line situations in which noise statistics are not known a priori. The idea is, either before or after updating, compute the innovation and use its square to get a handle on the noise level.

Proposition 8

$$\left\langle\left(y(k)-C(k)\tilde{x}(k)\right)\left(y(k)-C(k)\tilde{x}(k)\right)^H\right\rangle=C(k)\tilde{Q}(k)C(k)^H+V(k)$$

$$\left\langle\left(y(k)-C(k)\hat{x}(k)\right)\left(y(k)-C(k)\hat{x}(k)\right)^H\right\rangle=V(k)-C(k)\hat{Q}(k)C(k)^H \qquad (7\text{-}46)$$

$$=V(k)\left(C(k)\tilde{Q}(k)C(k)^H+V(k)\right)^{-1}V(k)$$

Proof:

The first expression is especially easy since the noise is independent of the previous updates and predictions:

$$\left\langle v(k)\left(x(k)-\tilde{x}(k)\right)^H\right\rangle=0\Rightarrow$$

$$\left\langle\left(y(k)-C(k)\tilde{x}(k)\right)\left(y(k)-C(k)\tilde{x}(k)\right)^H\right\rangle$$

$$=\left\langle\left(C(k)\left(x(k)-\tilde{x}(k)\right)+v(k)\right)\left(C(k)\left(x(k)-\tilde{x}(k)\right)+v(k)\right)^H\right\rangle$$

$$=C(k)\tilde{Q}(k)C(k)^H+V(k)$$

The second is similar but requires a little more care (use (7-45b)):

$$\left\langle (x(k)-\hat{x}(k))v(k)^H \right\rangle = \left\langle \left(x-\left(\tilde{x}+\hat{Q}C^H V^{-1}(Cx+v-C\tilde{x})\right)\right)v^H \right\rangle = -\hat{Q}C^H \Rightarrow$$

$$\left\langle (y(k)-C(k)\hat{x}(k))(y(k)-C(k)\hat{x}(k))^H \right\rangle$$

$$= \left\langle \left(C(k)(x(k)-\hat{x}(k))+v(k)\right)\left((x(k)-\hat{x}(k))^H C(k)^H + v(k)^H\right) \right\rangle$$

$$= C(k)\hat{Q}(k)C(k)^H - C(k)\left[\hat{Q}(k)C(k)^H\right] - C(k)\hat{Q}(k)C(k)^H + V(k)$$

$$= V(k) - C(k)\hat{Q}(k)C(k)^H$$

This then is expanded (dropping the index k) using (7-43a):

$$V - C\hat{Q}C^H = V - C\left(\tilde{Q}-\tilde{Q}C^H\left(C\tilde{Q}C^H+V\right)^{-1}C\tilde{Q}\right)C^H$$

$$= V - C\tilde{Q}C^H\left(I-\left(C\tilde{Q}C^H+V\right)^{-1}C\tilde{Q}C^H\right)$$

$$= \left(C\tilde{Q}C^H+V\right)\left(C\tilde{Q}C^H+V\right)^{-1}V - C\tilde{Q}C^H\left(C\tilde{Q}C^H+V\right)^{-1}\left(\left(C\tilde{Q}C^H+V\right)-C\tilde{Q}C^H\right)$$

$$= \left(C\tilde{Q}C^H+V\right)\left(C\tilde{Q}C^H+V\right)^{-1}V - C\tilde{Q}C^H\left(C\tilde{Q}C^H+V\right)^{-1}V$$

$$= V\left(C\tilde{Q}C^H+V\right)^{-1}V$$

Now, the next job is to specialize to the case with no dynamics $A(k) = I, W(k) = 0$ and the measurement noise is stationary at $V(k) = \sigma^2 I$. The filter is restated in simplified form, without explicitly using the noise intensity.

Proposition 9

Suppose $A(k) = I, W(k) = 0$ and the measurement noise is stationary at $V(k) = \sigma^2 I$. The filter can be restated as follows:

$$\begin{aligned} Q(k) &= \left(Q(k-1)^{-1} + C(k)C(k)^H\right)^{-1}, \quad Q(-1) \equiv \frac{1}{\sigma^2}Q_0 \\ \hat{x}(k) &= \hat{x}(k-1) + Q(k)C(k)^H\left(y(k)-C(k)\hat{x}(k-1)\right), \quad \hat{x}(-1) \equiv x_0 \end{aligned} \tag{7-47}$$

The noise estimate expressions (46) reduce to:

$$\begin{aligned} \left\langle (y(k)-C(k)\hat{x}(k-1))(y(k)-C(k)\hat{x}(k-1))^H \right\rangle &= \left(C(k)\hat{Q}(k-1)C(k)^H + I\right)\sigma^2 \\ \left\langle (y(k)-C(k)\hat{x}(k))(y(k)-C(k)\hat{x}(k))^H \right\rangle &= \left(I - C(k)\hat{Q}(k)C(k)^H\right)\sigma^2 \\ &= \left(C(k)\hat{Q}(k-1)C(k)^H + I\right)^{-1}\sigma^2 \end{aligned} \tag{7-48}$$

Proof:

The prediction step (7-44) is trivial $\tilde{x}(k) = \hat{x}(k-1), \tilde{Q}(k) = \hat{Q}(k-1)$. The update result follows easily by setting $\hat{Q}(k) = \sigma^2 Q(k)$. The results of (7-48) follow by these same substitutions.

These results are now applied to the problem of estimating a matrix that is multiplied by a time varying vector. For this we need the *vec* and *kronecker product* operations:

$$vec(A) \equiv \begin{bmatrix} a_{11} \\ \vdots \\ a_{n1} \\ \vdots \\ a_{1m} \\ \vdots \\ a_{nm} \end{bmatrix}, \quad A \otimes B \equiv \begin{bmatrix} a_{11}B & \cdots & a_{1m}B \\ \vdots & & \vdots \\ a_{n1}B & \cdots & a_{nm}B \end{bmatrix}$$

$$\begin{aligned} &vec(ABC) = (C^T \otimes A)vec(B) \\ &(A \otimes B)(D \otimes E) = (AD \otimes BE) \\ &(A \otimes B)^T = \left(A^T \otimes B^T\right) \\ &(A \otimes B)^{-1} = \left(A^{-1} \otimes B^{-1}\right) \end{aligned} \tag{7-49}$$

All matrices are complex in the above, and the identities shown work only where they are well defined (compatible dimensions, square inverses exist, etc.).

Proposition 10

Consider the simplified plant of Proposition 9 except that now the plant is given by:

$$\begin{aligned} &y(k) = X\ z(k) + v(k) \\ &\langle X \rangle = 0 \\ &\left\langle vec(X)vec(X)^H \right\rangle = \sigma^2\lambda^2 I \end{aligned} \tag{7-50}$$

with X an $ny \times nz$ complex matrix. The measurement noise has intensity $V(k) = \sigma^2 I$ as before.

Let:

$$\rho(k) \equiv 1 + z(k)^H \Phi(k-1) z(k) \tag{7-51}$$

Then the Kalman Filter can be expressed as:

$$\begin{aligned} &\Phi(k) = \left(\Phi(k-1)^{-1} + z(k)z(k)^H\right)^{-1}, \quad \Phi(-1) \equiv \lambda^2 I \\ &\hat{X}(k) = \hat{X}(k-1) + e(k)z(k)^H \Phi(k), \quad \hat{X}(-1) = 0 \\ &e(k) = y(k) - \hat{X}(k-1)z(k) = v(k) - \Delta X(k-1)z(k) \end{aligned} \tag{7-52}$$

This has the solution of:

$$\Phi(k) = \left(\lambda^{-2} I + \sum_{i=0}^{k} z(i) z(i)^H \right)^{-1}$$

$$\hat{X}(k) = \left[X \left(\sum_{i=0}^{k} z(i) z(i)^H \right) + \sum_{i=0}^{k} v(i) z(i)^H \right] \Phi(k)$$

$$\Delta X(k) \equiv \hat{X}(k) - X = -\lambda^{-2} X \Phi(k) + \left(\sum_{i=0}^{k} v(i) z(i)^H \right) \Phi(k) \qquad (7\text{-}53)$$

and for $v(i) \equiv 0$,

$$\Delta X(k) \Phi(k)^{-1} \Delta X(k)^H$$

$$= \lambda^{-4} X \Phi(k) X^H = \Delta X(k-1) \Phi(k-1)^{-1} \Delta X(k-1)^H - \frac{e(k) e(k)^H}{\rho(k)}$$

Hence in the no noise case if $\lambda^2 >> \left\| \left(\sum_{i=0}^{k} z(i) z(i)^H \right)^{-1} \right\|$ (and the inverse exists), then $\hat{X}(k) \equiv X$.

This can occur after nz steps. Also, the norm of the error $\Delta X(k) \Phi(k) \Delta X(k)^H$ and its trace are monotonically non-decreasing. Further, $e(k) / \sqrt{\rho(k)}$ is an L_2 sequence. If $z(k)$ is bounded, then also $e(k) = -\Delta X(k-1) z(k)$ is an L_2 sequence. Finally, the rows of $\hat{X}(k)$ are linear combinations of the $z(i)^H, i = 0, \cdots, k$ even when noise is present.

The noise level can be estimated using:

$$\left\langle \left(y(k) - \hat{X}(k-1) z(k) \right)^H \left(y(k) - \hat{X}(k-1) z(k) \right) \right\rangle = ny \rho(k) \sigma^2$$

$$\left\langle \left(y(k) - \hat{X}(k) z(k) \right)^H \left(y(k) - \hat{X}(k) z(k) \right) \right\rangle = ny \frac{\sigma^2}{\rho(k)} \qquad (7\text{-}54)$$

Proof:

Rewrite (7-50a) in terms of the *vec* of X:

$$y(k) = \left(z(k)^T \otimes I \right) vec(X) + v(k)$$

Then, identify equivalent elements with (7-41b):

$$x = vec(X)$$

$$C(k) = \left(z(k)^T \otimes I \right)$$

Expand the expressions for the filter Q:

$$Q(-1) \equiv \frac{1}{\sigma^2} Q_0 = \lambda^2 I$$

$$Q(k) = \left(Q(k-1)^{-1} + C(k)C(k)^H\right)^{-1} = \left(Q(k-1)^{-1} + \left(z(k)^T \otimes I_{ny}\right)\left(z(k)^T \otimes I_{ny}\right)^H\right)^{-1}$$

$$= \left(Q(k-1)^{-1} + \left(z(k)^T z(k)^* \otimes I_{ny}\right)\right)^{-1}$$

We note that the dimension of Q is $nynz \times nynz$, and that it begins with and maintains a kronecker product structure, which, on comparison with (7-52a) is just:

$$Q(k) = \Phi(k)^* \otimes I_{ny}$$

This is easily proven by induction if desired. Using this new expression for Q we can plug in for the weight update of (7-47b):

$$\hat{x}(k) = \hat{x}(k-1) + Q(k)C(k)^H\left(y(k) - C(k)\hat{x}(k-1)\right)$$

$$= \hat{x}(k-1) + \left(\Phi(k)^* \otimes I\right)\left(z(k)^T \otimes I\right)^H\left(y(k) - \left(z(k)^T \otimes I\right)\hat{x}(k-1)\right)$$

$$= \hat{x}(k-1) + \left(\Phi(k)^* z(k)^* \otimes I\right)\left(y(k) - \hat{X}(k-1)z(k)\right)$$

$$\Downarrow$$

$$\hat{X}(k) = \hat{X}(k-1) + \left(y(k) - \hat{X}(k-1)z(k)\right)z(k)^H\Phi(k)$$

The formula (7-53a) is easily proven by checking its validity for $k = -1$ and then inductively for k given the result for k-1. The second formula (7-53b) is then shown as follows:

$$\hat{X}(-1) = 0$$

$$\hat{X}(k) = \hat{X}(k-1) + \left(y(k) - \hat{X}(k-1)z(k)\right)z(k)^H\Phi(k)$$

$$= \hat{X}(k-1) + \left(Xz(k) + v(k) - \hat{X}(k-1)z(k)\right)z(k)^H\Phi(k)$$

$$= \hat{X}(k-1)\left(\Phi(k)^{-1} - z(k)z(k)^H\right)\Phi(k) + Xz(k)z(k)^H\Phi(k) + v(k)z(k)^H\Phi(k)$$

$$= \left(X\left(\sum_{i=0}^{k-1} z(i)z(i)^H\right) + \sum_{i=0}^{k-1} v(i)z(i)^H\right)\left(\lambda^{-2}I + \sum_{i=0}^{k-1} z(i)z(i)^H\right)^{-1}$$

$$\times\left(\lambda^{-2}I + \sum_{i=0}^{k} z(i)z(i)^H - z(k)z(k)^H\right)\Phi(k) + Xz(k)z(k)^H\Phi(k) + v(k)z(k)^H\Phi(k)$$

$$= \left(X\left(\sum_{i=0}^{k-1} z(i)z(i)^H\right) + \sum_{i=0}^{k-1} v(i)z(i)^H\right)\Phi(k) + Xz(k)z(k)^H\Phi(k) + v(k)z(k)^H\Phi(k)$$

$$= \left(X\left(\sum_{i=0}^{k} z(i)z(i)^H\right) + \sum_{i=0}^{k} v(i)z(i)^H\right)\Phi(k).$$

The observations about the case when $\lambda^{2} >> \left\| \left(\sum_{i=0}^{k} z(i)z(i)^{H} \right)^{-1} \right\|$ are obvious from the expressions and the size of *z(i)*. The error expressions can be shown as follows:

$$\Delta X(k) \equiv \hat{X}(k) - X$$

$$= X\left(\sum_{i=0}^{k} z(i)z(i)^{H} \right)\left(\lambda^{-2} I + \sum_{i=0}^{k} z(i)z(i)^{H} \right)^{-1} - X + \left(\sum_{i=0}^{k} v(i)z(i)^{H} \right)\Phi(k)$$

$$= X\left\{ \left(\sum_{i=0}^{k} z(i)z(i)^{H} \right) - \left(\lambda^{-2} I + \sum_{i=0}^{k} z(i)z(i)^{H} \right) \right\}\Phi(k) + \left(\sum_{i=0}^{k} v(i)z(i)^{H} \right)\Phi(k).$$

$$= -\lambda^{-2} X\Phi(k) + \left(\sum_{i=0}^{k} v(i)z(i)^{H} \right)\Phi(k) = -\lambda^{-2} X\Phi(k) + \left(\sum_{i=0}^{k} v(i)z(i)^{H} \right)\Phi(k)$$

$$v(k) = 0 \quad \Rightarrow \quad \Delta X(k)\Phi(k)^{-1}\Delta X(k)^{H} = \lambda^{-4} X\Phi(k)X^{H}$$

$$= \lambda^{-4} X\left(\Phi(k-1)^{-1} + z(k)z(k)^{H} \right)^{-1} X^{H}$$

$$= \lambda^{-4} X\left(\Phi(k-1) - \Phi(k-1)z(k)\left(1 + z(k)^{H}\Phi(k-1)z(k)\right)^{-1} z(k)^{H}\Phi(k-1) \right) X^{H}$$

$$= \lambda^{-4} X\Phi(k-1)X^{H} - \frac{\lambda^{-4} X\Phi(k-1)z(k)z(k)^{H}\Phi(k-1)X^{H}}{\rho(k)}$$

$$= \Delta X(k-1)\Phi(k-1)^{-1}\Delta X(k-1)^{H} - \frac{\Delta X(k-1)z(k)z(k)^{H}\Delta X(k-1)^{H}}{\rho(k)}$$

$$= \Delta X(k-1)\Phi(k-1)^{-1}\Delta X(k-1)^{H} - \frac{e(k)e(k)^{H}}{\rho(k)}$$

The fact that the rows of $\hat{X}(k)$ are linear combinations of the $z(i)^H, i = 0,\cdots,k$ can be shown by using the rule:

$$A^H\left(aI + AA^H\right)^{-1} = A^H\left(\tfrac{1}{a}I - \tfrac{1}{a}A\left(I + \tfrac{1}{a}A^HA\right)^{-1}\tfrac{1}{a}A^H\right)$$

$$= \left(I - \tfrac{1}{a}A^HA\left(I + \tfrac{1}{a}A^HA\right)^{-1}\right)\tfrac{1}{a}A^H = \left(\left(I + \tfrac{1}{a}A^HA\right) - \tfrac{1}{a}A^HA\right)\left(I + \tfrac{1}{a}A^HA\right)^{-1}\tfrac{1}{a}A^H$$

$$= \left(I + \tfrac{1}{a}A^HA\right)^{-1}\tfrac{1}{a}A^H$$

$$A^H\left(aI + AA^H\right)^{-1} = \left(aI + A^HA\right)^{-1}A^H$$

applied to:

$$Z(k) \equiv \left[z(0) \quad \cdots \quad z(k)\right]$$

$$V(k) \equiv \left[v(0) \quad \cdots \quad v(k)\right]$$

$$\hat{X}(k) = \left[X\left(\sum_{i=0}^{k} z(i)z(i)^H\right) + \sum_{i=0}^{k} v(i)z(i)^H\right]\Phi(k)$$

$$= \left[XZ(k)Z(k)^H + V(k)Z(k)^H\right]\left(\lambda^2 + Z(k)Z(k)^H\right)^{-1}$$

$$= \left[XZ(k) + V(k)\right]Z(k)^H\left(\lambda^2 + Z(k)Z(k)^H\right)^{-1}$$

$$\hat{X}(k) = \left\{\left(XZ(k) + V(k)\right)\left(\lambda^2 + Z(k)^HZ(k)\right)^{-1}\right\}Z(k)^H$$

The statements about the convergence of the sequences are shown as follows:

$$tr\left\{\Delta X(k)\Phi(k)^{-1}\Delta X(k)^H\right\} =$$

$$= tr\left\{\Delta X(k-1)\Phi(k-1)^{-1}\Delta X(k-1)^H\right\} - \frac{\|e(k)\|^2}{\rho(k)}$$

$$= tr\left\{X\Phi(-1)^{-1}X^H\right\} - \sum_{i=0}^{k}\frac{\|e(i)\|^2}{\rho(i)} \geq 0$$

$$\Rightarrow \sum_{i=0}^{k}\frac{\|e(i)\|^2}{\rho(i)} \leq tr\left\{X\Phi(-1)^{-1}X^H\right\} \quad \forall k \geq 0 \Rightarrow \frac{\|e(k)\|}{\sqrt{\rho(k)}} \text{ is } L_2.$$

Then if $z(k)$ is bounded,

$$\|z(k)\| \leq \bar{z} \Rightarrow \rho(k) = 1 + z(k)^H\Phi(k-1)z(k) \leq 1 + \bar{z}^2\|\Phi(k-1)\| \leq 1 + \bar{z}^2\|\Phi(-1)\|$$

$$\Rightarrow \frac{\|e(k)\|}{\sqrt{1 + \bar{z}^2\|\Phi(-1)\|}} \leq \frac{\|e(k)\|}{\sqrt{\rho(k)}} \Rightarrow \|e(k)\| = \|\Delta X(k-1)z(k)\| \text{ is } L_2.$$

The noise expression (7-54a) is shown as follows, starting from (7-48):

$$tr\left\langle \left(y(k)-C(k)\hat{x}(k-1)\right)\left(y(k)-C(k)\hat{x}(k-1)\right)^{H}\right\rangle = tr\left(C(k)\hat{Q}(k-1)C(k)^{H}+I\right)\sigma^{2}$$

$$\left\langle \left\|y(k)-C(k)\hat{x}(k-1)\right\|^{2}\right\rangle = tr\left(\left(z(k)^{T}\otimes I\right)\left(\Phi(k)^{*}\otimes I\right)\left(z(k)^{*}\otimes I\right)+I\right)\sigma^{2}$$

$$=\left(1+z(k)^{T}\Phi(k)^{*}z(k)^{*}\right)tr\left(I_{ny}\right)\sigma^{2}=\left(1+z(k)^{H}\Phi(k)z(k)\right)ny\sigma^{2}$$

This last step follows from the fact that Φ is positive definite and any expression $\mu^{H}\Phi\mu$ is therefore real (and equal to its conjugate). The other expression for the noise is derived analogously.

This last proposition allows the estimation of matrices, keeping track of all algebraic structure of previous measurements, while using a normalized covariance matrix of order only $nz \times nz$. This is a vital economy that makes the application of this algorithm for weight estimation practical, especially when combined with the square root filtering formulae developed in [9].

7.4 Adaptive Cancellation Algorithms

This section gives the mathematical statements of the ANC algorithms that were used during experiments on the ASTREX testbed. The algorithms will be stated in terms of the plant equation (8), shown here with an additional measurement noise *v(k)*:

$$\begin{aligned} &y(k)=G+Pu(k)+v(k) \\ &y(k),G,v(k)\in\mathbf{C}^{ny},\quad u(k)\in\mathbf{C}^{nu},\quad P\in\mathbf{C}^{ny\times nu} \end{aligned} \tag{7-55}$$

7.4.1 High Pass Perturbation Algorithm

This algorithm makes use of the fact that *G* is a constant, so that a high pass filter applied to (7-53) leaves only the *Pu(k)* portion of the equation, and some noise. First we define the algorithm, and then explore its behavior.

Definition of High Pass Perturbation Algorithm

Real Constants:

$\lambda \in (0,1)$	low pass filter constant ($\frac{1}{ny(1+nu)}$ typical)
$\alpha \in (0,1]$	weight update constant (1 typical)
$u_{\max}$	maximum magnitude of any control component
$q_{\max}, q_o > 0$	perturbation constants ($q_{\max}=\frac{u_{\max}}{4}$, $q_o=\frac{q_{\max}}{100}$ typical)
$d(r)$	column $(r+1)$ of identity matrix I_{nu}

Initialize:

$$\begin{aligned} r(0) &= 0 \\ u_{low}(0) &= u(0) \\ y_{low}(0) &= y(0) \\ u(1) &= q_o d(0) \end{aligned} \tag{7-56}$$

Low and High Pass Filters of y and u:

$$\begin{aligned} y_{low}(k) &= y_{low}(k-1) + \lambda\left(y(k) - y_{low}(k-1)\right) \\ y_{hi}(k) &= y(k) - y_{low}(k) \\ u_{low}(k) &= u_{low}(k-1) + \lambda\left(y(k) - u_{low}(k-1)\right) \\ u_{hi}(k) &= y(k) - u_{low}(k) \end{aligned} \tag{7-57}$$

Plant Estimation:

$$\begin{aligned} e_P(k) &= y_{hi}(k) - \hat{P}(k-1)u_{hi}(k) \\ \hat{P}(k) &= \hat{P}(k-1) + \frac{\alpha}{\left\|u_{hi}(k)\right\|^2} e_P(k)u_{hi}(k)^H \\ e_G(k) &= y_{low}(k) - \hat{P}(k)u_{low}(k) - \hat{G}(k-1) \\ \hat{G}(k) &= \hat{G}(k-1) + \alpha\, e_G(k) \end{aligned} \tag{7-58}$$

Perturbation Update:

$$\begin{aligned} \eta(k) &= \frac{1}{nu}\left\|\hat{P}(k)\right\|_F^2 \\ q(k) &= \begin{cases} -q(k-1) & k \text{ even or } \eta(k) = 0 \\ \min\left(q_{\max}, 4\left|q(k-1)\right|, \sqrt{\dfrac{\left\|e_P(k)\right\|^2}{\eta(k)} + \dfrac{q_o^{\,2} q(k)^2}{q_o^{\,2} + q(k)^2}}\right) & \text{else} \end{cases} \\ r(k) &= \begin{cases} r(k-1) & k \text{ odd} \\ \mathrm{mod}\left(r(k-1)+1, nu\right) & \text{else} \end{cases} \end{aligned} \tag{7-59}$$

Control Computation:

$$u(k) = \mathrm{limiter}\left(-P(k)^+ G(k), u_{\max} - q_{\max}\right) + q(k)d(r(k)) \tag{7-60}$$

The notation $P(k)^+$ means the pseudo inverse. The limiter scales its first argument such that each element's magnitude is less than or equal to the second argument.

There are several interesting things that can be said about this algorithm. The analysis will all be done assuming no noise ($v(k) \equiv 0$). The first step is to show that the filtering splits up the plant into two pieces.

Proposition 11

$$y_{hi}(k) = Pu_{hi}(k)$$
$$y_{low}(k) = G + Pu_{low}(k) \qquad (7\text{-}61)$$

Proof:

The low pass filter operator is linear, so that

$$y_{low}(k) = G_{low}(k) + Pu_{low}(k),$$
$$G_{low}(k) = G_{low}(k-1) + \lambda\left(G - G_{low}(k-1)\right), \quad G_{low}(0) = G$$
$$\Rightarrow G_{low}(k) = G \Rightarrow y_{low}(k) = G + Pu_{low}(k).$$

But then, by definition,

$$y_{hi}(k) \equiv y(k) - y_{low}(k) = \left(G + Pu(k)\right) - \left(G + Pu_{low}(k)\right)$$
$$= P\left(u(k) - u_{low}(k)\right) = Pu_{hi}(k).$$

We now show that the High Pass Perturbation algorithm always converges to the best canceling control in the absence of noise.

Proposition 12

In the absence of measurement noise, the High Pass Perturbation algorithm produces a control history u(k) that converges to the optimum canceling control $-P^{+}G$. Further, $\hat{P}(k) \to P, \hat{G}(k) \to G$.

Proof:

The first goal is to show that the $\hat{P}(k)$ estimator is of the form of equation (7-33) and that its input is "sufficiently rich" to force $\hat{P}(k) \to P$. The first of these steps is easy, the second, very difficult.

That the $\hat{P}(k)$ estimator of (7-58a,b) is in the proper form of equation (7-33) is established readily by setting:

$$x = vec(P),\ \hat{x}(k) = vec(\hat{P}(k)),\ C(k) = u_{hi}(k)^{T} \otimes I_{ny},\ s(k) = \frac{\sqrt{\alpha}}{\left\|u_{hi}(k)\right\|}$$

Application of Proposition 3 then ensures that $\hat{P}(k)$ converges to a fixed value, and that $\left\|s(k)e(k)\right\| = \dfrac{\sqrt{\alpha}}{\left\|u_{hi}(k)\right\|}\left\|e_P(k)\right\|$ is an L_2 sequence convergent to zero.

We now argue heuristically that the control input $u_{hi}(k)$ is sufficiently rich. Equation (7-60) shows that the control is composed of the canceling estimate + a perturbation used to stimulate learning. Note that the perturbative part consists of a scaling factor *q(k)* times the periodic pattern *d(r(k))*. The pattern of the perturbation simply marches through the columns of an

identity matrix on every other iteration, thus ensuring that the entire space of u is spanned. Furthermore, the arrangement of probing first in one direction and then in the opposite direction on the next step with the same magnitude ensures that this attribute is carried through to $u_{hi}(k)$. This suffices to ensure that $\hat{P}(k) \rightarrow P$.

Now, the G estimation system is from equations (7-58c,d) and (7-61b):

$$y_{low}(k) = G + Pu_{low}(k)$$

$$e_G(k) = y_{low}(k) - \hat{P}(k)u_{low}(k) - \hat{G}(k-1)$$

$$= G - \hat{G}(k-1) + \left(P - \hat{P}(k)\right)u_{low}(k) \rightarrow G - \hat{G}(k-1)$$

$$\hat{G}(k) = \hat{G}(k-1) + \alpha\, e_G(k) \rightarrow \hat{G}(k-1) + \alpha\left(G - \hat{G}(k-1)\right)$$

The asymptotic equation for the weight $\hat{G}(k)$ ensures that it arrives at G. Then the control achieves its goal $-P^{+}G$ + the perturbation term. That the perturbation term goes to zero is seen from the following argument.

First, recall that $\frac{\sqrt{\alpha}}{\|u_{hi}(k)\|}\|e_P(k)\|$ converges to zero. But, the limiting of the control output ensures that $\|u_{hi}(k)\|$ is bounded by a constant, say K. However,

$$\frac{\sqrt{\alpha}}{K}\|e_P(k)\| \leq \frac{\sqrt{\alpha}}{\|u_{hi}(k)\|}\|e_P(k)\|$$

ensuring that $\|e_P(k)\|$ itself goes to zero. Then, reviewing equation (7-59) reveals that asymptotically the perturbation amplitude *q(k)* will obey equation (7-39) of Proposition 5 on every other step, so that it decays to zero. Thus the perturbation gradually disappears.

This ends the proof.

The conditions of equation (7-61) are reestablished after an actuator or sensor failure after several time constants of the low pass filter, λ^{-1}, have elapsed. Then the same argument again shows that the weights are perfectly estimated in the no noise case.

This algorithm has the disadvantage of requiring several time constants λ^{-1} and α^{-1} to converge. The next algorithm is as quick as possible, and also has better noise filtering capabilities.

7.4.2 Kalman Filter Cancellation Algorithm

The algorithm is specified first, and then we do some analysis.

Definition of Kalman Filter Cancellation Algorithm

Real Constants:

$\lambda \in (0,1)$	low pass filter constant ($\frac{1}{(1+nu)}$ typical)
$u_{\max}$	maximum magnitude of any component of the canceling control
$r_{\max}, r_o > 0$	perturbation constants ($r_{\max} = \frac{u_{\max}}{9}$, $r_o = 0.1$ typical)
ν	plant change detection factor (20 typical)
β^2	magnitude of initial Q matrix (10^8 typical)
$d(s)$	column $(s+1)$ of matrix $\left[I_{nu} \quad -I_{nu} \quad 0_{nu\times 1}\right]$

Initialize:

$$r(-1) = r_0, \quad s(-1) = 2nu, \quad \hat{\sigma}^2(-1) = 0, \quad ii(-1) = 2nu+1, \quad u(-1) = 0$$
$$Q(-1) = \beta^{-2} I, \quad \hat{P}(-1) = 0, \quad \hat{G}(-1) = 0 \qquad (7\text{-}62)$$

Innovation and uncertainty ratio computation:

$$e(k) = y(k) - \hat{P}(k-1)u(k) - \hat{G}(k-1)$$
$$\eta(k) = \begin{bmatrix} 1 \\ u(k) \end{bmatrix} \qquad (7\text{-}63)$$
$$\rho(k) = 1 + \eta(k)^H Q(k-1)\eta(k)$$

Plant Change Detection and Noise Level Estimation:

$$\tilde{\sigma}^{\,2}(k) = \frac{\|e(k)\|^2}{ny\,\rho(k)}$$

$$\text{plant\_changed} = \left(\tilde{\sigma}^{\,2}(k) > \nu\,\hat{\sigma}^{\,2}(k-1)\right) \wedge (ii = 0)$$

if (plant_changed)

$$Q(k) = \left(\beta^{-2} I + \eta(k)\eta(k)^H\right)^{-1}$$

$$ii = 2nu + 1$$

$$\chi(k) = 0$$

else

$$Q(k) = \left(Q(k-1)^{-1} + \eta(k)\eta(k)^H\right)^{-1}$$

$$ii = \max(ii - 1, 0)$$

$$\chi(k) = \min\left(1, 3\frac{2\sqrt{\tilde{\sigma}^{\,2}(k)\hat{\sigma}^{\,2}(k-1)}}{\rho(k)\left(\tilde{\sigma}^{\,2}(k) + \hat{\sigma}^{\,2}(k-1)\right)}\right)$$

endif

$$\hat{\sigma}^{\,2}(k) = \hat{\sigma}^{\,2}(k-1) + \lambda\left(\tilde{\sigma}^{\,2}(k) - \hat{\sigma}^{\,2}(k-1)\right) \tag{7-64}$$

$$\chi(k) = \min\left(1, 3\frac{2\sqrt{\tilde{\sigma}^{\,2}(k)\hat{\sigma}^{\,2}(k-1)}}{\rho(k)\left(\tilde{\sigma}^{\,2}(k) + \hat{\sigma}^{\,2}(k-1)\right)}\right)$$

Plant Estimation:

$$\bar{\eta}(k) = Q(k)\eta(k)$$

$$\hat{G}(k) = \hat{G}(k-1) + e(k)\bar{\eta}_1(k)^* \tag{7-65}$$

$$\hat{P}(k) = \hat{P}(k-1) + e(k)\bar{\eta}_{2:(nu+1)}(k)^H$$

Perturbation Update:

$$r(k) = \left(1 - \chi(k)\right)r_{\max} + \chi(k)\ \min\left(\frac{\|e(k)\|}{\phi + \left\|\hat{P}(k)\right\|}, 10r(k-1), r_{\max}\right) \tag{7-66}$$

$$s(k) = \text{mod}\left(s(k) + 1, 2nu + 1\right)$$

Control Computation:

$$u(k+1) = \chi(k)\,\text{limiter}\left(-P(k)^+ G(k), u_{\max}\right) + r(k)d(s(k)) \tag{7-67}$$

The limiter scales its first argument such that each element's magnitude is less than or equal to the second argument.

Before proceeding formally to the convergence results and analysis, we should cover a few points about the algorithm to aid in understanding its functionality.

- The variable $\chi(k)$ is a heuristic measure of the degree of convergence or "confidence factor" in the estimation results. It is used to prevent applying control based on only a few noisy measurements. It varies between 0 and 1, 0 representing no confidence, and 1, full confidence in the estimates. Before convergence it is a factor times an index showing the volatility of the noise estimate, $\frac{2\sqrt{\tilde{\sigma}^2(k)\hat{\sigma}^2(k-1)}}{\left(\tilde{\sigma}^2(k)+\hat{\sigma}^2(k-1)\right)}$, divided by the noise adjustment factor $\rho(k)$ from equation (53) of Proposition 10. This is used in the perturbation adjustment of (66) and in the control computation of (67).
- A plant change is detected if the measurement noise estimate based on the current measurement is very much larger than the average computed from filtering previous estimates (equation 64). In addition, to detect a plant change requires that a certain number of iterations have passed since the last plant change, as measured using the counter *ii*. Upon detecting a plant change, the normalized covariance $Q(k)$ is set to a large value and the confidence factor $\chi(k)$ is set to zero.

Now we turn to the main proposition concerning this algorithm, which proves that Proposition 10 applies to this algorithm until a plant change is detected.

Proposition 13

Proposition 10 applies to the Kalman Filter Cancellation Algorithm with

$$X=\begin{bmatrix}G & P\end{bmatrix},\quad \hat{X}(k)=\begin{bmatrix}\hat{G}(k) & \hat{P}(k)\end{bmatrix},\quad z(k)=\begin{bmatrix}1\\ u(k)\end{bmatrix},\quad \lambda=\beta$$

until a plant change is detected. Furthermore,

$$\sum_{i=0}^{k} z(i)z(i)^H = Z(k)Z(k)^H,\quad Z(k)=\begin{bmatrix}z(0) & \cdots & z(k)\end{bmatrix}$$

has full rank for $k \geq nu$ until a plant change is detected, no earlier than $k = 2nu+1$.

Proof:

It is apparent that the estimation part of the algorithm meets the conditions of Proposition 10 with the assignments given.

We now show that the inputs $z(k), k = 0,1,\cdots,nu$ form a complete $nu+1$ dimensional basis.

First, in Proposition 10, the rows of the weight estimate $\hat{X}(k)$ are linear combinations of the input vectors $z(i)^H, i = 0,1,\cdots,k$.

Thus, the $\hat{P}(k)$ have rows proportional to the $u(k)^H$ vectors input:

$$\hat{X}(k) = (XZ+V)\left(\lambda^{-2}I + Z^H Z\right)^{-1} Z^H$$

$$\left[\hat{G}(k) \quad \hat{P}(k)\right] = (XZ+V)\left(\lambda^{-2}I + Z^H Z\right)^{-1} \left[\begin{array}{c|c} 1 & u(0)^H \\ \vdots & \vdots \\ 1 & u(k)^H \end{array}\right]$$

$$\Rightarrow \hat{P}(k) = (XZ+V)\left(\lambda^{-2}I + Z^H Z\right)^{-1} \begin{bmatrix} u(0)^H \\ \vdots \\ u(k)^H \end{bmatrix}$$

Then the canceling control estimate $-\hat{P}(k)^+ \hat{G}(k)$ can only have columns proportional to the control inputs up to that time, because the pseudo inverse preserves the spaces of its argument (its columns space is the row space of its argument, and visa versa). Note that the limiter in the control computation only scales this estimate without changing its direction. Therefore, the control is composed of the scaled canceling control estimate, which is a linear combination of the control inputs specified so far, and the perturbation term, which on examination is seen to add one new actuator to the scheme on each iteration:

$s(k) = k$

$d(s(k)) = column\ k+1\ of\ I_{nu}$

$\Downarrow$

$$U \equiv \begin{bmatrix} u(0) & u(1) & u(2) & \cdots & u(nu) \end{bmatrix} = \begin{bmatrix} 0 & r(0) & * & \cdots & * \\ 0 & 0 & r(1) & \cdots & * \\ 0 & 0 & 0 & \ddots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & r(nu-1) \end{bmatrix}$$

$$Z = \begin{bmatrix} z(0) & z(1) & z(2) & \cdots & z(nu) \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 1 \\ & U & \end{bmatrix} = \text{upper triangular}$$

The * elements correspond to the canceling control estimate at each step.

Note that the formula for $r(k)$ ensures that it cannot be zero for k in this range, because the value of $\chi(k)$ is guaranteed to be small:

$$\chi(k) \equiv \min\left(1, 3\frac{2\sqrt{\tilde{\sigma}^{\,2}(k)\hat{\sigma}^{\,2}(k-1)}}{\rho(k)\left(\tilde{\sigma}^{\,2}(k)+\hat{\sigma}^{\,2}(k-1)\right)}\right) \le \frac{3}{\rho(k)}$$

$$\rho(k) \equiv 1 + z(k)^H \Phi(k-1) z(k) = 1 + \begin{bmatrix} * \\ r(k-1) \\ 0 \end{bmatrix}^H \begin{bmatrix} * & 0 & 0 \\ 0 & \lambda^{-2} & 0 \\ 0 & 0 & \lambda^{-2} I \end{bmatrix}^{-1} \begin{bmatrix} * \\ r(k-1) \\ 0 \end{bmatrix}$$

$$= 1 + \|*\|^2 + r(k-1)^2 \lambda^{\,2} \ge r(k-1)^2 \lambda^{\,2}$$

$$\Rightarrow \chi(k) \le \frac{3}{r(k-1)^2 \lambda^{\,2}}$$

Where the * represents arbitrary values due to previous inputs. The initial $r(-1) = r_0$ is certainly not small enough to make $\chi(k)$ appreciable. Then reference to (66) shows that the perturbation $r(k)$ is near its maximum value for the times of interest.

Thus, *Z(nu)* is of full rank as required. It will remain so as long as no plant change is detected.

The plant change is not detected unless the noise estimate (which is unbiased according to Proposition 10) experiences a large boost which lies several times outside of the expected variation of the estimate based on stationary Gaussian statistics. Also, it can not be too near a previous plant change.

The "confidence factor" $\chi(k)$ is fairly tolerant of variations in the estimated noise level, as long as they are not orders of magnitude changes, such as occur at the very start of identification, due to the arbitrary nature of the initial covariance choice.

The Kalman Filter estimation algorithm exhibits the fastest possible convergence rate, because, as shown in Proposition 10 of Subsection 7.3.2, the exact answer is in principle obtainable in nu + 1 steps of the algorithm, the number of steps required just to excite all available degrees of freedom in the system! The reason this is possible is because of the retention of the "covariance" matrix in the algorithm, which amounts to maintaining second-derivative information in the optimization problem, not just gradient information as occurs, for instance, in the LMS algorithm.

8. Experiments and Results

In this chapter we discuss the various experiments run with the algorithms described in Chapter 7.

8.1 ANC Trip 1 Experiments and Results

The first ANC demonstration trip took place from Jan 8 to Jan 11, 1996. The trip was dedicated to the testing of the Hi-Pass Perturbation Algorithm described in Subsection 7.4.1. As mentioned in Chapter 2 two motion sensors were used during this test visit: linear accelerometers mounted in the X and Y directions on the secondary tower. Tower LPACTs 2 and 3 were used as disturbance sensors, and all six ACESA strut bending channels were used as cancellation actuators.

Three tones were introduced at structural modal frequencies so as to cause the greatest possible excitation in the tower. These were at 10.85, 12.3, and 13.9 Hz.

Open loop results are shown below in Figure 8-1. These are the power spectral densities (PSDs) of the voltages, after amplification, of the two accelerometers mounted on the tower.

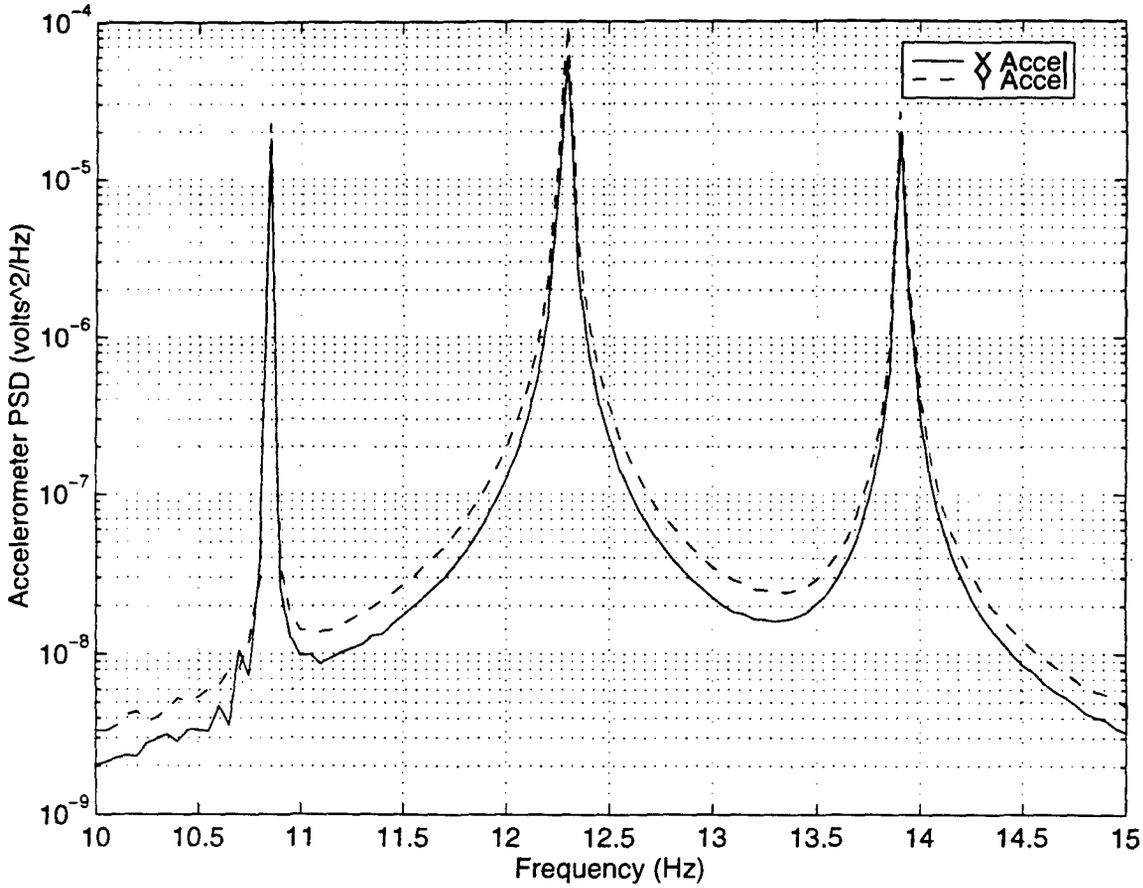


Figure 8-1. Open Loop Accelerometer PSDs for Trip 1

It should be noted that the FFT measurements during this visit were made using a "boxcar" window on the samples. Because the signals are sines and cosines, with significant correlation over long times, this windowing technique caused large "skirts" around the major tones, which swamped out the noise at frequencies where no disturbance was present. When the dominant tones were canceled, this caused an apparent drop in the "noise floor," since the skirts from the large tones were reduced significantly along with their parent tones. This should be kept in mind when reviewing the results from this first trip presented here. (Testing during the second visit used "Hanning" windowing, which removes this effect.)

The algorithm was started with no prior system information and with all six cancellation actuator channels functioning. The algorithm was deemed to have converged when all demodulated tone measurements of the motion sensors were below 50 mV, the approximate noise floor at the A/D converters. This required about 5 minutes, and produced the results shown below in Figure 8-2. All closed loop results are the sum of the relevant x and y PSDs.

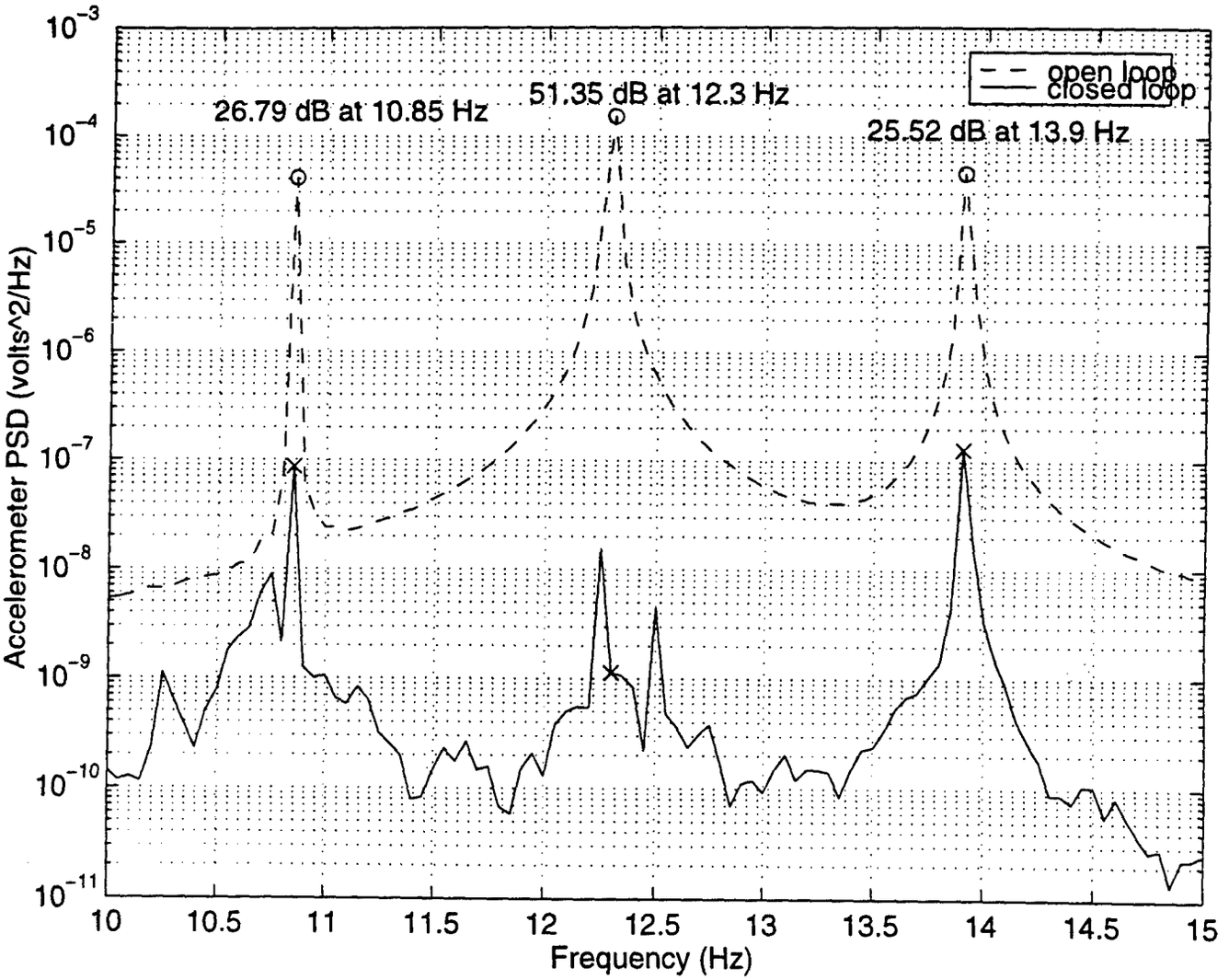


Figure 8-2. Closed Loop Accelerometer PSDs for Trip 1, All Actuators On

The controller achieved 27 dB on the first tone, 51 dB on the second, and 26 dB on the third tone. This was low enough that other tone components were also present in the closed loop.

The next step was to turn off an actuator channel, actuator 3, which corresponded to the vertical bending axis of strut 2 (at the 4:00 position). Convergence to a new solution required about 15 minutes and produced the result shown below in Figure 8-3.

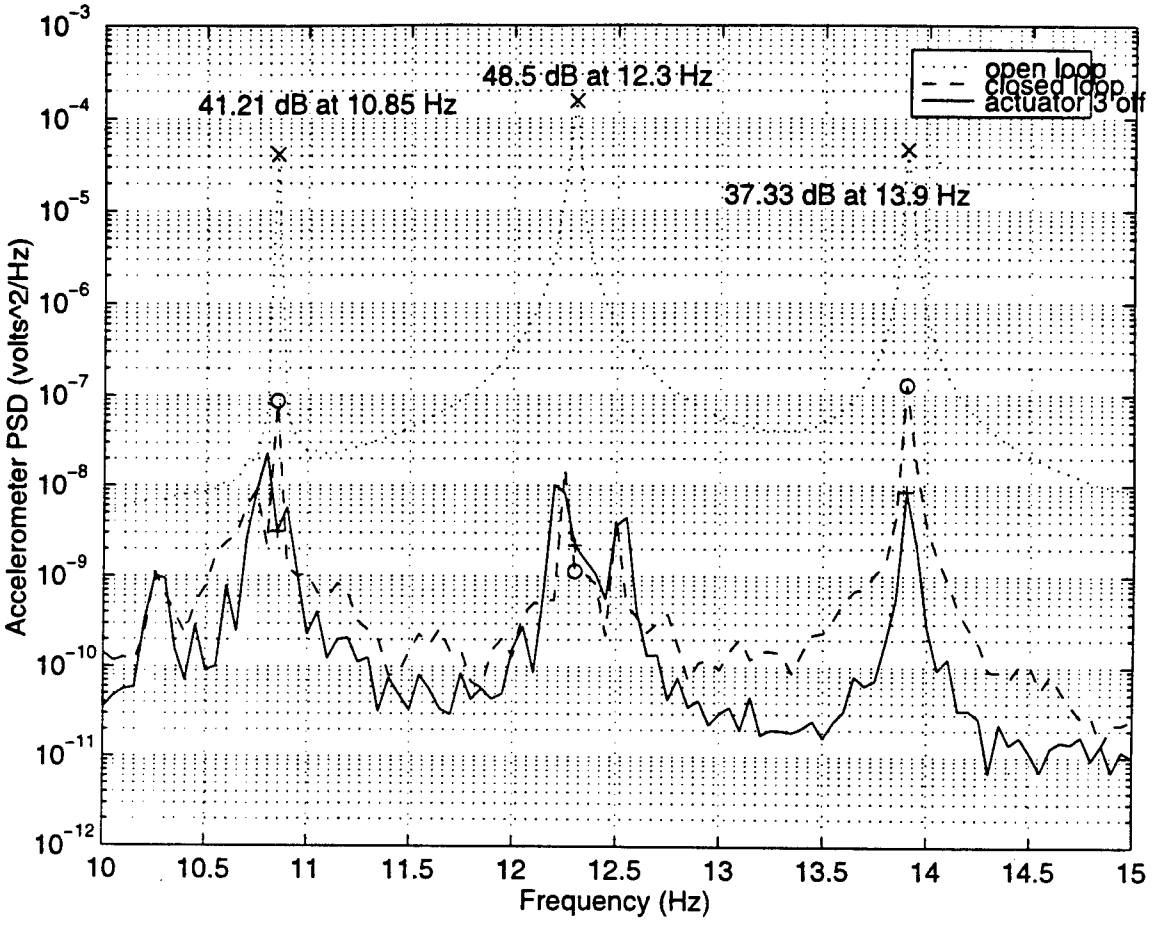


Figure 8-3. Closed Loop Accelerometer PSDs for Trip 1, Actuator 3 Off

The results were slightly improved over the case when of all actuators working. The first and third tones each saw about 10 dB improvement, and the second tone remained about the same. It is difficult to say why this was so, except that a longer period was allowed for the convergence of all tones, so that tones 1 and 3 converged more precisely while waiting for tone 2. At least it is apparent that actuator 3 was not, in itself, required for good performance.

The next step was to turn off actuator 6, corresponding to the vertical bending axis of the 8:00 ACESA strut. This required about 10 minutes to converge to the results shown below in Figure 8-4.

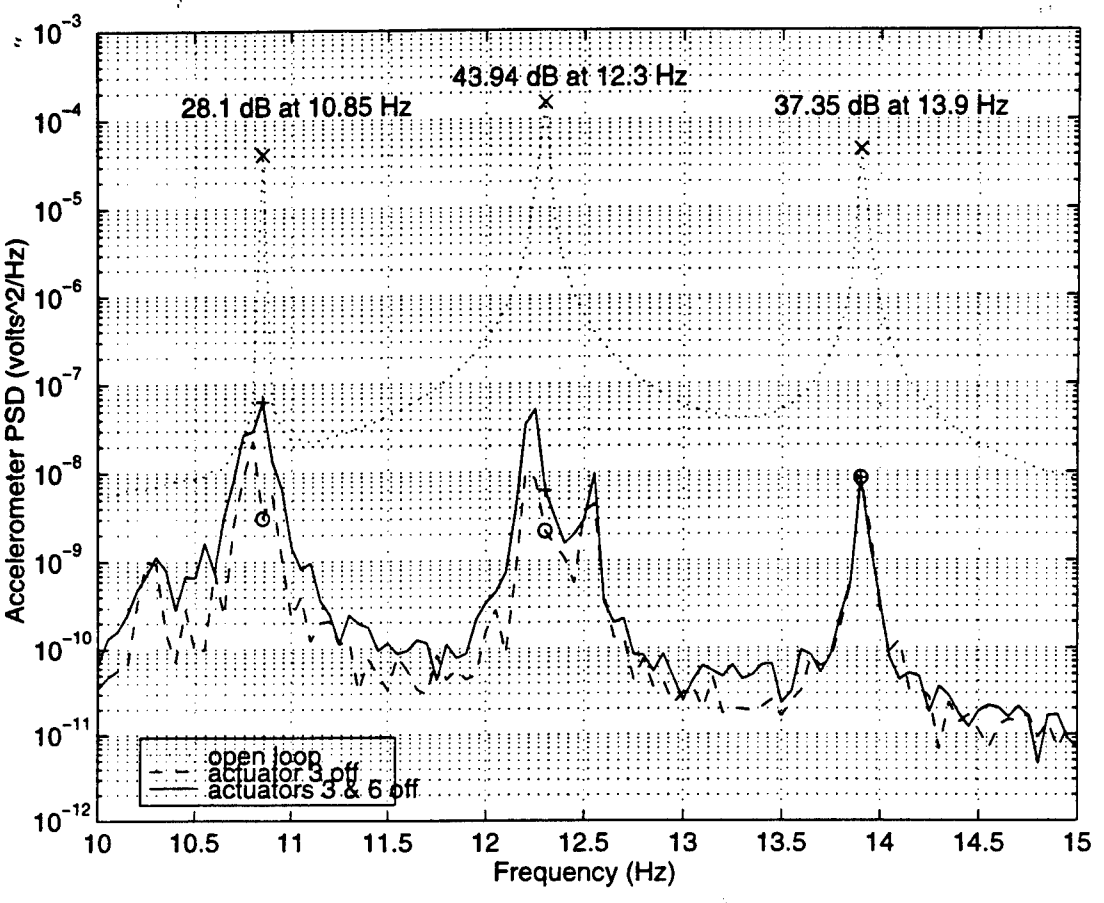


Figure 8-4. Closed Loop Accelerometer PSDs for Trip 1, Actuators 3 & 6 Off

The results were slightly degraded from the previous case, by about 10 dB on the first tone and 5 dB on the second.

The most extreme failure case came next, when actuators 2 and 5 were also turned off, corresponding to the horizontal axis of strut 1 and the vertical axis of strut 3. Thus, only actuators 1 and 4 were left on, controlling the vertical axis of strut 1 and the horizontal axis of strut 2. Convergence was imperfect due to lack of sufficient control authority. After 30 minutes the result was as shown below in Figure 8-5.

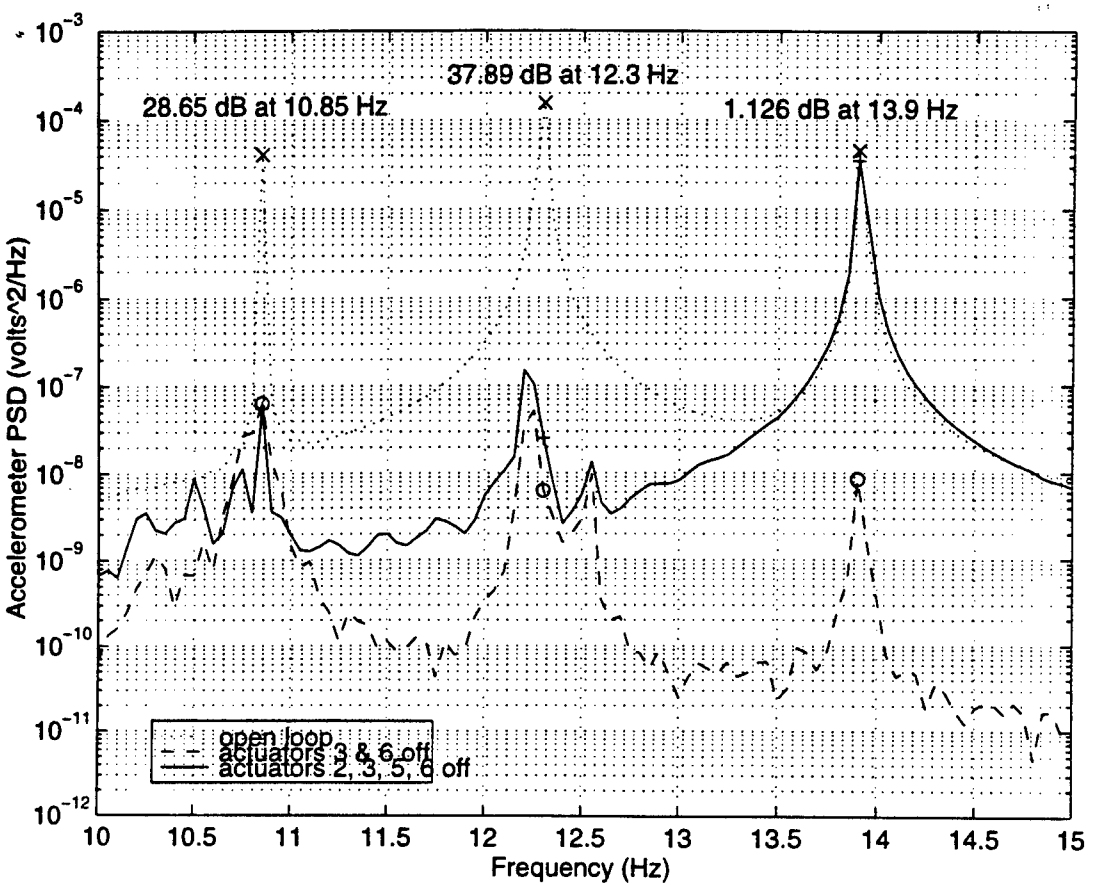


Figure 8-5. Closed Loop Accelerometer PSDs for Trip 1, Only Actuators 1 & 4 On

Tones 1 and 2 were still controlled down to approximately the same level as previously. Tone 3, however, was virtually at its open loop state. Apparently, only the third tone had insufficient authority to control with the available actuators.

The results of this first trip were very good. However, there were three major improvements recommended:

- Raise the disturbance levels so that visitors could see the performance improvements more readily. This required some changes in the amplifier gains to accommodate larger signal levels.
- Create a visual, laser-based display of performance by mounting light sources on the primary mirror assembly and reflecting them off of the secondary and back to the primary.
- Improve convergence speed if possible.

These improvements were made and demonstrated on the second, final trip.

8.2 ANC Trip 2 Experiments and Results

The second ANC demonstration trip took place from Jun 10 to Jun 14, 1996. The trip was dedicated to the testing of the Kalman Filter Cancellation Algorithm described in Subsection 7.4.2. As mentioned in Chapter 6, three motion sensors were used during this test visit: angular velocity sensors mounted in the X, Y, and Z directions on the secondary tower. Tower LPACTs 2 and 3 were used as disturbance sensors, and all six ACESA strut bending channels were used as cancellation actuators.

The three tonal frequencies were different from those used in the previous trip because there was too little output from the ACESA struts to cancel the angular motion when the input disturbance was turned up high enough to make a good demonstration. Transfer functions were measured and a new set of frequencies were chosen at 10.15, 10.625, and 12.725 Hz.

Open loop results are shown below in Figure 8-6. These are the power spectral densities (PSDs) of the voltages, after amplification, of the three angular velocity sensors mounted on the tower.

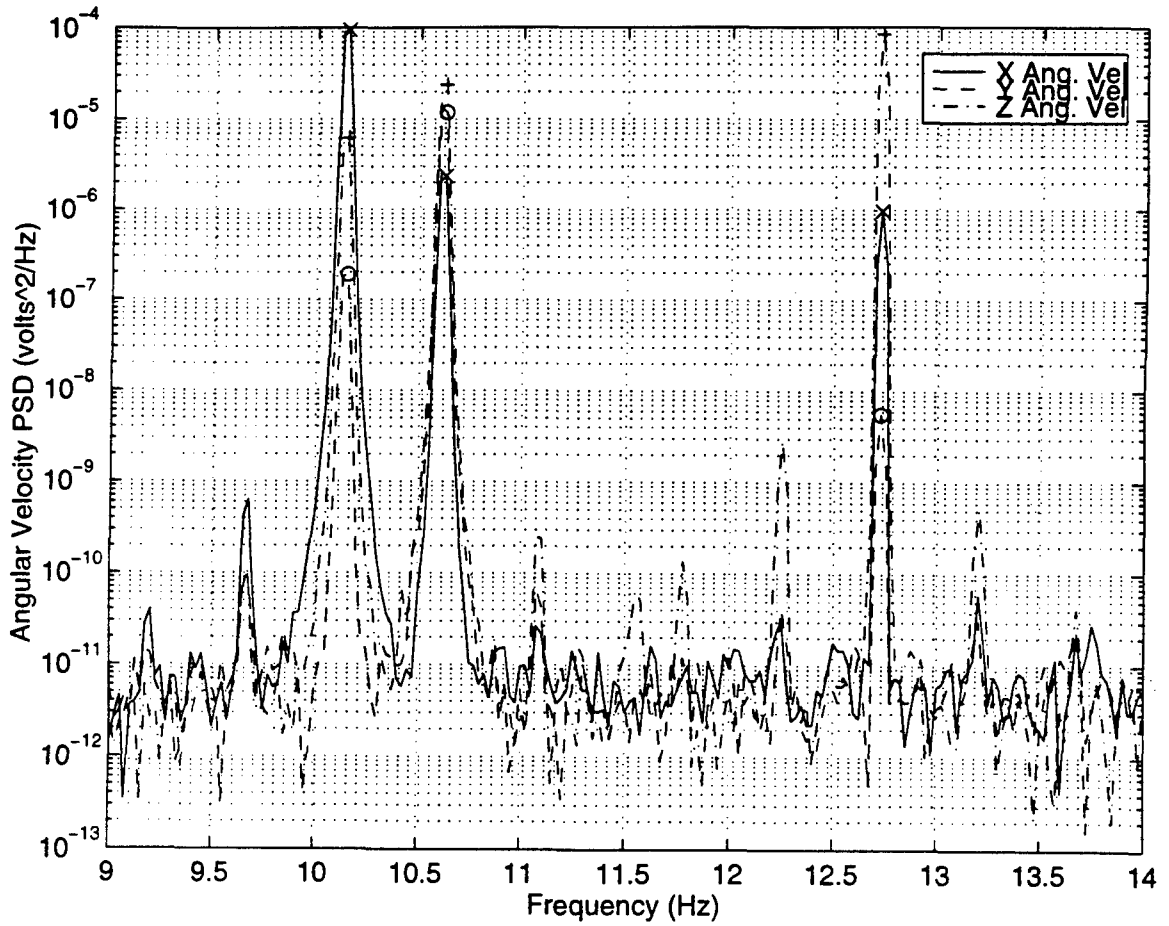


Figure 8-6. Open Loop Accelerometer PSDs for Trip 2

The additional spurs at, say, 12.25 Hz are due to small nonlinearities in the disturbance system, which cause mixing of the frequencies.

The Kalman Filter Cancellation Algorithm was found to converge in approximately 7 minutes, as opposed to approximately 10 minutes for the Hi-Pass Perturbation Algorithm. The results for the case when all actuators are working is shown below in Figure 8-7. As before, all closed loop results are shown as the sum of the PSDs of the separate motion sensors.

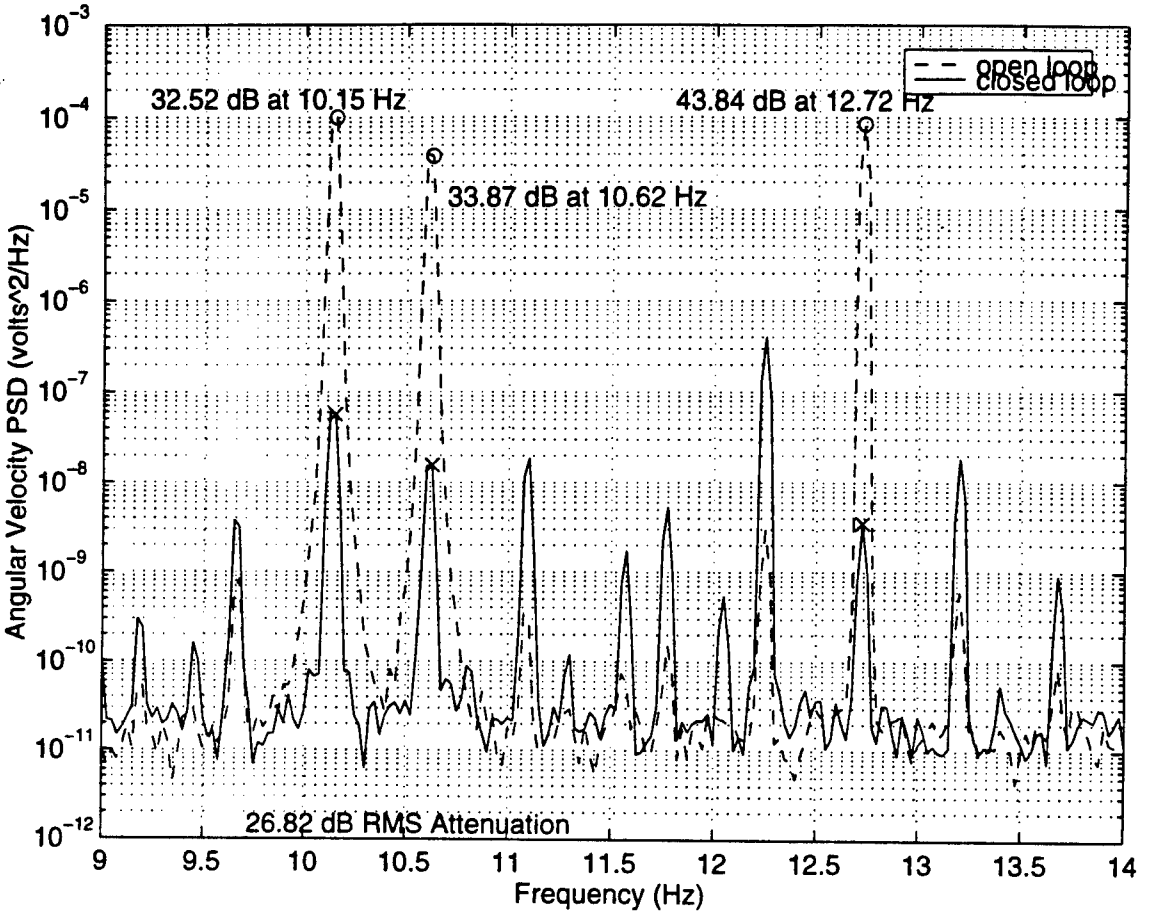


Figure 8-7. Closed Loop Angular Velocity PSD for Trip 2

The algorithm achieved excellent cancellation of the tones: 33 dB at 10.15 Hz, 34 dB at 10.62 Hz, and 44 dB at 12.72 Hz. However, the difference frequency "spurs" have been increased significantly, for example at 12.25 Hz. An investigation was undertaken to determine the cause of this effect. It was found that the commands to the actuators contained no such spurs, and that the sensors all showed them, including the accelerometers. Evidently, there is a nonlinearity in the actuation system which causes intermodulation products of the three tones to be generated. Even with these additional spurs, the RMS attenuation across the 9-14 Hz band was 27 dB.

The next step was to turn off actuator 3. The new algorithm immediately detected a change in the plant and reconverged in 7 minutes. This is a major improvement over the Hi-Pass Perturbation Algorithm, which required considerably longer to converge after a failure than it did initially. The results for the case of actuator 3 turned off are shown below in Figure 8-8.

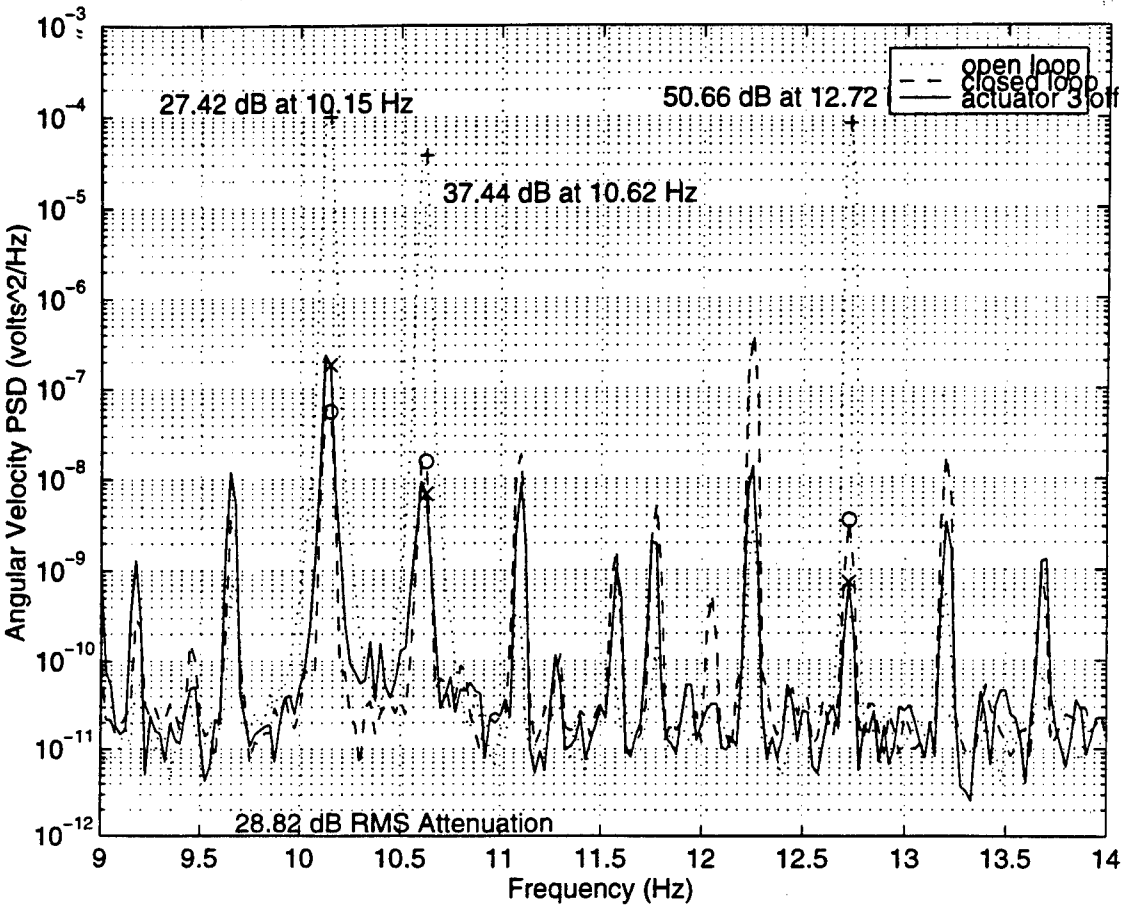


Figure 8-8. Closed Loop Angular Velocity PSD for Trip 2, Actuator 3 Off

This case had better over all performance than the case when actuator 3 was turned on (except at 10.15 Hz), as did the similar case during the first trip. Note also that the spur at 12.25 Hz is lower by about 10 dB. This would tend to suggest that Actuator 3 was the source of a significant portion of the nonlinearity.

The last case is with both actuator 3 and actuator 6 turned off. Once again, convergence required about 7 minutes. The results are shown below in Figure 8-9.

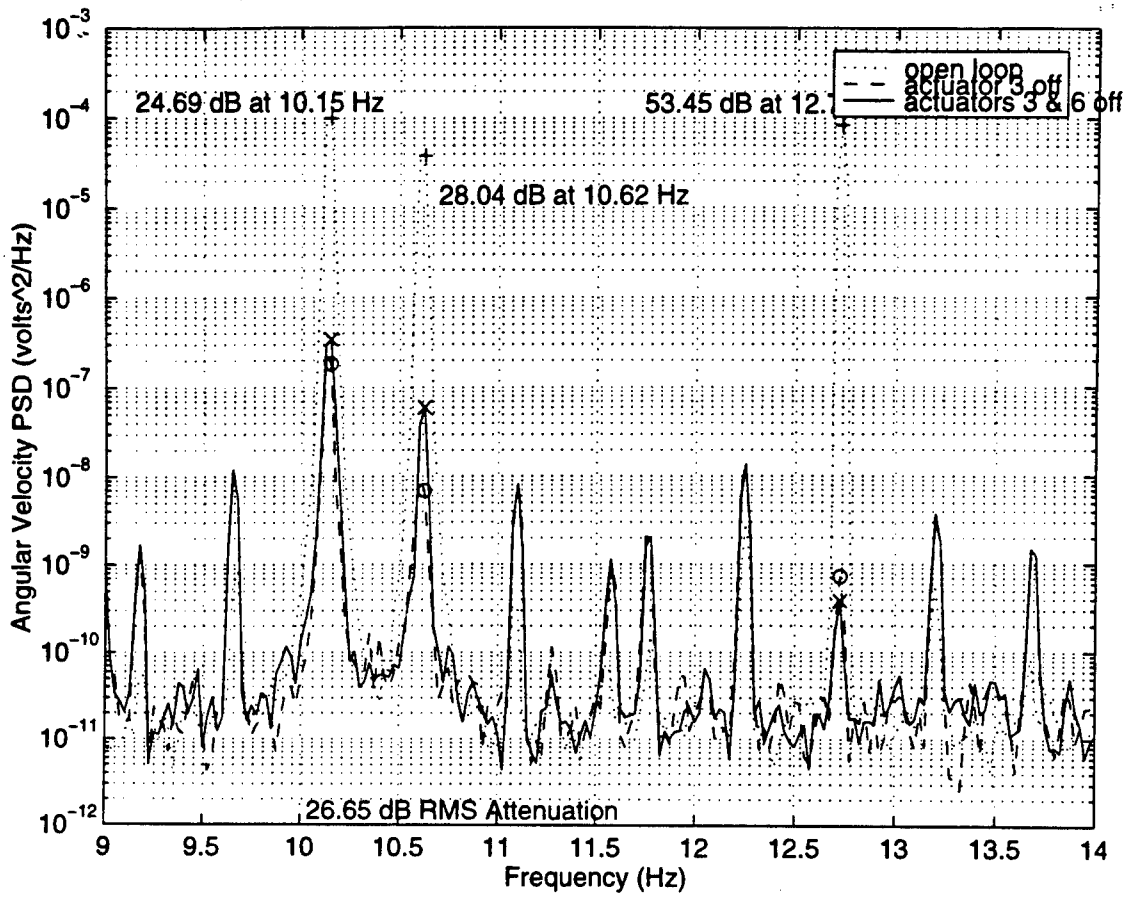


Figure 8-9. Closed Loop Angular Velocity PSD for Trip 2, Actuators 3 & 6 Off

Total attenuation was slightly worse than for the case with only actuator 3 turned off, but still slightly better than the full closed-loop case. Note that the spurs remain at approximately the same level as in the previous case, indicating that actuator 6 did not contribute much to the nonlinearity.

These canceling control values were saved in a restart program as initial conditions, to allow quick demonstration without the need for 7 minutes of adaptation. The response of the system as can be judged by observing the moving laser spots was similar to turning off the disturbance suddenly. The system response damps out naturally over about 10-20 seconds, due to the low damping inherent in the structure. Of course, the cancellation system is feedforward in nature and does not add any damping to the open loop structure. Instead, it injects a signal that counteracts the effects of the disturbance at the sensor locations.

The ANC system was adept at accommodating any actuator failure, and required no modeling data to operate whatsoever.

9. ANC Final Software Configuration

Before discussing the construction of the software, we list the procedure for running the demo.

9.1 Demonstration Procedure

1. Ensure that the ASTREX testbed is floated. Ensure all actuators and sensors are powered on, enabled, and operational. Ensure the three tonal disturbances are turned on, and that the sync signals are connected to the ANC system. Ensure the lasers for visual performance monitoring are turned on. Turn on the power to the ANC system, and allow it to warm up for approx. 5 minutes.
2. Hit carriage return several times on the user terminal, until the system is logged in and the OS-9 prompt appears.
3. Type "demo" and hit carriage return. A menu of options for control are listed, as shown below in Table 9-1. Ensure the hardware reflects the choice chosen. If one of the restart options (1-3) is chosen, the program uses the saved solution to start, and then continues to learn. Additional actuators can be switched on or off after waiting about 1½ minutes. Reconvergence after a configuration change requires about 7 minutes.
4. Hit control-C when you wish to exit. The actuator commands are zeroed out. The computer may require up to 20 seconds to respond to your control-C command.

Table 9-1. Menu Choices for the Demo Program

Menu Choice	Description
1 -- All Actuators Working	Restart from a saved solution with all actuators working. This gives immediate results, and then continues to learn. Be sure all actuators are turned on when this is started.
2 -- Actuator 3 Failed	Restart from a saved solution with actuator 3 turned off. This gives immediate results, and then continues to learn. Be sure all actuators except actuator 3 are turned on when this is started.
3 -- Actuators 3 & 6 Failed	Restart from a saved solution with actuators 3 and 6 turned off. This gives immediate results, and then continues to learn. Be sure all actuators except actuators 3 and 6 are turned on when this is started.
4 -- Start Learning from Scratch	This starts learning from scratch using any configuration of actuators on. It requires about 7 minutes to converge.
5 -- Exit	Leave the program before starting.

9.2 Software Hierarchy and Overview

The ANC software is arranged in the C code structure shown below in Figure 9-1.

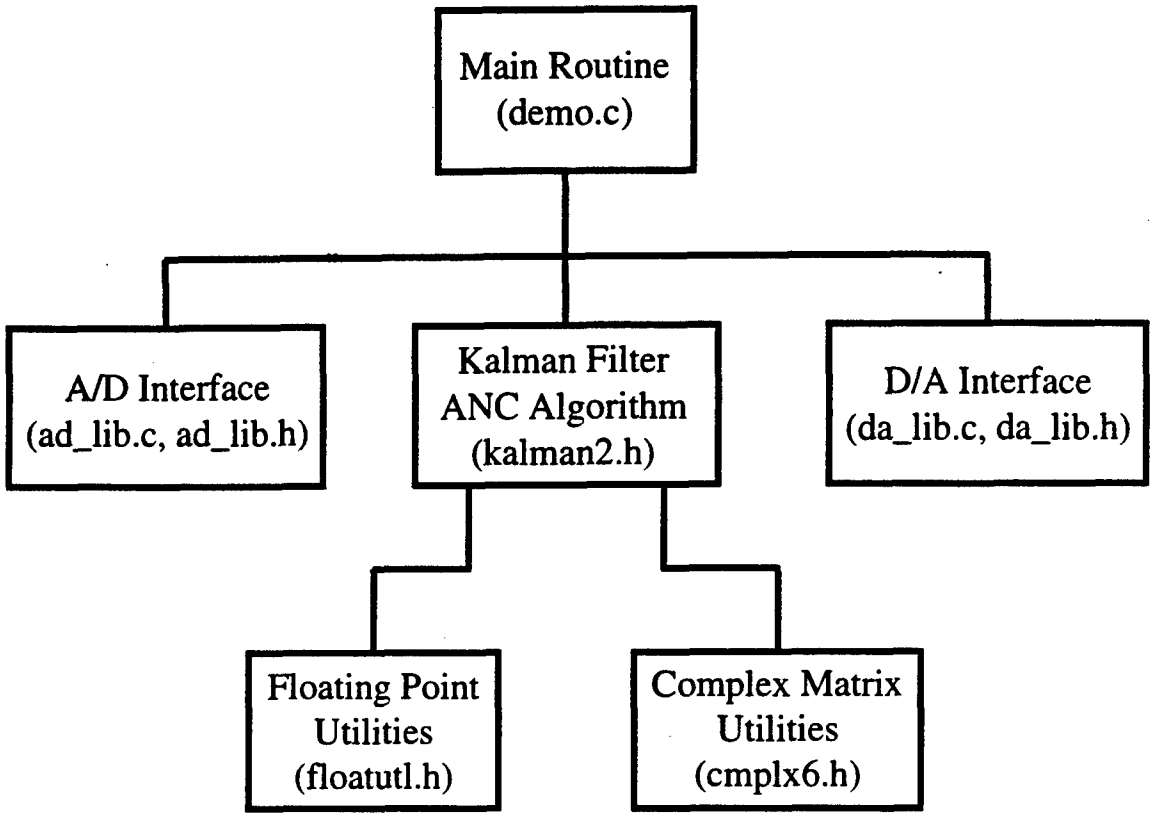


Figure 9-1 Software Hierarchy

The structure is reflected in the file names. The main routine is called "demo.c". It handles the interface with the A/D and the D/A converters and the timing of the control loop. The ANC algorithm itself resides in "kalman2.h". It makes use of complex matrix routines and floating point routines found in "floatutl.h" and "cmplx6.h".

We will cover this structure by first discussing the main routine, then the complex and floating point utilities, in preparation for the discussion of the algorithm itself.

Throughout the software frequent use is made of data types that are pointers to structures. The software itself is found in Appendix A, and should be referenced in the following discussions.

9.3 Main Routine (demo.c)

The code for "demo.c" is arranged in distinct sections. The first section consists of the "include" file references. These are self-explanatory references given the hierarchy of Figure 9-1.

The second section consists of declarations of important system constants and file names. The constant CHANS is the maximum number of channels that will ever be accessed for either the D/A or the A/D converters. Normally, the number would be 36 (6 actuators x 3 tones x 2

numbers (real/imaginary) per tone). However, two D/A channels had to be skipped because of designed-in firmware/hardware problems.

The third code section consists of macros that are useful for manipulating or displaying data. The SORT and UNSORT macros map the physical channels of the hardware onto the ordered pairs of real numbers that the control algorithm expects. The data for this mapping is stored in the file "addamap.dat". It should not be modified without due consideration, since the wiring of the A/D and D/A converter connectors is tied to this map. The other macros are for displaying the state of the control process.

The fourth section is a little routine for handling a control-C that is typed by the user. The "intercept(handle_ctlC);" statement in the main routine tells the operating system to call the handler when the control-C is asserted. The routine sets the flag if the correct signal is asserted, which informs the main loop that it should quit.

The fifth and final code section is the main routine itself. It begins with variable declarations, including the controllers which are of type "controller". There is actually an array of three controller objects, each with its own internal state.

Next, the A/D and D/A map is read in. Also, the table of A/D offset biases that were previously measured is read in. The A/D and D/A hardware is initialized. Then the controllers are initialized.

Next, the user is asked what option he would like to execute. His answer is recorded as a string in the variable "ans". If the answer involved using data recorded previously, the appropriate file is opened, and the restart data is read up using the controller routine written for that purpose.

The control-C handler is installed.

At last, the control loop begins. The data is retrieved and sorted from the A/D converter. The biases are subtracted from the readings. Then the control algorithms are called and the results are printed. The data is sorted for distribution to the D/A converters, and then sent. Finally, the controller waits the appointed time before returning to the top of the loop.

When the control-C is detected, the loop exits and the end code is executed. First, the D/A converters are zeroed out. Then, the controller states are saved in the file "demo.dat". This file is a restart file that could be copied over one of the files listed at the top of the code, if it corresponded with the correct configuration. This would be necessary if the plant were to change and the best cancellation results were required immediately, as for a "demo".

We now move on to the low level routines used for implementing complex matrix arithmetic.

9.4 Complex Matrix Routines and LDL^H Decomposition (cmplx6.h)

Complex numbers are an integral part of the mathematics of the cancellation algorithm. Unfortunately, since C does not implement complex numbers as an intrinsic part of the language, they had to be implemented using data structures. They were implemented here using a structured data type:

```
typedef struct
{  float real;
   float imag;
```

```
} *complex;
```

Thus, a complex number is a pointer to a structure with a real and an imaginary part. A complex number X must be allocated and deallocated, and its real and imaginary parts are accessed as X->real or X->imag.

All the usual operations are defined in the include file, whether as macros or as functions.

Complex matrices are also defined:

```
typedef struct
{  int nrows;
   int ncols;
   complex ptr;
} *complex_matrix;
```

Clearly, a complex_matrix M must be allocated and deallocated. Its dimensions are accessed as M->nrows, M->ncols. A special macro is used to point to a particular complex element:

```
#define cm(z,i,k) ( ((z)->ptr) + (i) + (k)*((z)->nrows) )
```

so that cm(M,i,k) is the complex number M(i,k). Remember that indices in C start with 0!

The final data structure defined in this include file is the LDL^H decomposition. This is a numerically stable way of representing Hermitian positive semi-definite matrices.

$$W = W^H \geq 0 \quad \Rightarrow \quad W = LDL^H,$$

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ L_{21} & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ L_{n1} & L_{n2} & \cdots & 1 \end{bmatrix}, \quad D = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{bmatrix}, \quad d_k \geq 0$$

This is represented here by the following data structure:

```
typedef struct
{  int n;
   complex Lptr;
   float *Dptr;
} *LDL_demod;
```

so the L and D matrices are not represented as matrix structures, but as arrays of the right kind of element. If Q is an LDL decomposition, then an element (i,k) of L is accessed using a macro as LDL_L(Q,i,k). Similarly, the ith element of D is accessed as LDL_D(Q,i). Special routines are used for computing things like $Q^{-1}z$ to ensure that the best economies and accuracy are realized.

9.5 Control Algorithm (kalman2.h)

The control algorithm resides in this file. It actually is treated as an object, with its own internal data structure and routines for allocation, deallocation, and iteration.

The data structure is:

```
typedef struct
{  int ny, nu, ipert, arm;
   float lambda, sig2, r, maxu, maxr;
   complex_matrix Phat, Phatinv, Ghat, e, um, ym, eta,
                  Qeta, Qetap, Qetag;
    LDL_decomp Qinv, Qinvold;
} *controller;
```

The algorithm itself is in the routine "control_iterate(y,u,c)", where c is a variable of type controller. It closely follows the definition of Chapter 3, with the following variable identifications:

Software	Algorithm Definition
ym	$y(k)$
um	$u(k)$
Qinv	$Q(k)^{-1}$
ipert	$s(k)$
INITIAL_PERT	r_o
(direct manipulation of perturbed actuator)	$d(s(k))$
arm	$ii(k)$
TOLER_CTL	β^{-2}

The code is intended to be commented well enough that it is relatively easy to compare with the algorithm definition.

The remaining routines are just for allocation, deallocation, display, or reading from and writing to files.

9.6 A/D Converter Interface (ad_lib.c, ad_lib.h)

This is a separately compiled library for low level to access to the 601 A/D converter card, which also controls the A/D multiplexer boards by means of the 601's channel selector. The include file just has prototypes for the routines that are intended to be called by users.

The A/D board has its own on board computer. It is sent commands in ASCII via the "DPR" area. One sets the board up with these commands and then starts it off. The board then provides the data in the DPR area at the address you have told it. Experimentation was required with delays at certain points in the code to ensure that the system reset occurred as desired, reliably.

Several routines just display the contents of the two status registers.

No interrupts are used in this code.

This code should not be altered without close study of the manual and the existing code.

9.7 D/A Converter Interface (da_lib.c, da_lib.h)

This is a separately compiled library for low level to access to the 641 D/A converter cards. The include file just has prototypes for the routines that are intended to be called by users. Each D/A converter is separately accessed and sampled with its own routine. This was a brute force way of ensuring that the user did not have to know bus addresses, etc. The code is much shorter and more straightforward than the A/D code, because there is no on-board computer to contend with.

Once one has studied the hardware manual a little, the code should be fairly self-explanatory.

No interrupts are used in this code.

10. ANC Final Hardware Configuration

In this chapter we briefly describe the commercial hardware that was bought for the ANC system, and cover in somewhat greater detail the operation of the custom analog circuit cards. For ease of reference, Figure 6-4 is repeated below as Figure 10-1. In the figure, the green signals are analog, and all others are digital.

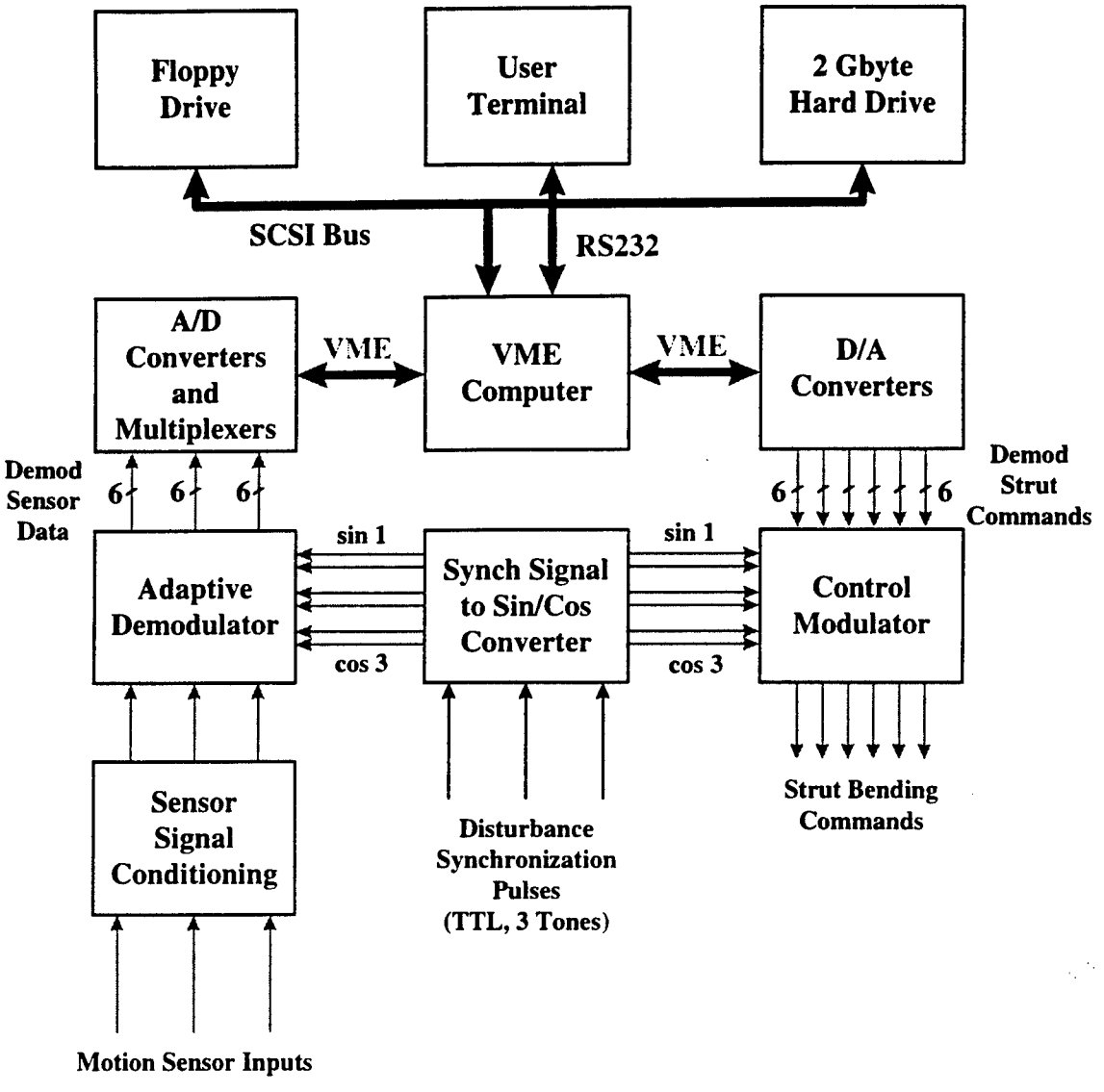


Figure 10-1. Block Diagram of the ANC System

The VME computer and peripherals, the A/D converters and multiplexers, and the D/A converters are all commercial off-the-shelf components. The Adaptive Demodulator and the Control Modulator are found on one type of custom analog card (the processing for one actuator and one sensor on each card), while the sensor signal conditioning and the sin/cos converter are mounted on the large custom analog card. Signals are passed between the analog cards on the

VME backplane (unused lines). The Demod Sensor Data and the Demod Strut Commands are passed on the D-connector terminated custom wire bundles on the front of the VME cage.

We proceed with a discussion of the hardware from left to right in the VME card cage.

10.1 VME Computer and Peripherals

The VME computer is the left-most card in the card cage. It is the bus-master of the VME bus. It is a VME167 single board computer manufactured by Motorola. It interfaces with the A/D and D/A converters via the VME bus. It interfaces with the peripherals through a small Motorola interface card on the back of the card cage, which plugs into the VME bus at the back behind the single board computer.

Both disk drives, the floppy and the large NPI hard drive, are connected on the same SCSI bus. This interface is somewhat unusual for floppy drives and makes replacement of this item more difficult than just going to the nearest computer store.

The NPI drive is the boot device. It has all of the software for the OS-9 real-time operating system (RTOS) on board. No fancy RTOS features such as semaphores, etc., are used in the ANC system.

The floppy drive can be used either in the PC (FAT) format, or in the OS-9 format. The PC format is accessed using device /px. The OS-9 format is accessed using the /d0 device. Remember to use the "dos" command to convert and copy files to or from /px, or else the carriage return/line feed sequence will not be right.

Programming was done using the somewhat troublesome µMACS editor. "Make" files were executed to compile and link, e.g. "make -f=demo.m" to recompile and link the main program with include files.

10.2 A/D Converter and Multiplexers

The A/D converter works in concert with two multiplexer boards. The A/D board is a Datel DVME-601B type with an ADC-12/2A module. The multiplexer boards are two Datel DVME-641 boards. The A/D converter is 12 bit, bipolar, with an input range of ±10 volts.

The A/D converter is operated in the differential mode because this is the only mode which supports multiplexers. The multiplexer boards themselves, however, are operated in single-ended mode. Analog signals are only input into the multiplexer boards. Each multiplexer board handles up to 32 channels of input. Thus, 64 channels of input are available on the two multiplexer boards. Of these, only the first 36 are hooked up to the analog electronics, 32 channels on the first multiplexer board, and 4 on the second. These appear as channels 0-35 to the user of the A/D interface routines.

The 601 has an onboard Motorola MC68010 processor. Commands from the VME167 host computer are written into a dual-port memory area called the DPR. Data is also passed through this area.

The slave multiplexer 641 boards are accessed automatically when the channel register on the 601 board addresses a higher channel than is available on its own board. Communication

between the 641 and the 601 occur using the top connectors on the front of the cards, not the VME bus. The 641 multiplexers only take power from the VME bus.

10.3 D/A Converters

The D/A converters are Datel 622 analog output boards, with 12 bits resolution over a range of ±10 volts (this is selectable). There are 16 channels per card for a total of 48 possible channels accessible. Of these, channel 8 (numbering from 0) on each card is unusable due to a vendor design problem. The controller actually requires only 36 channels. Therefore, the channel ranges accessed are 0-7, 9-15 on each of the first two cards, and 0-5 on the third card.

10.4 Analog Demodulation Cards

The function and operation of the analog demodulator from an analytical point of view were discussed at some length in Section 7.2. At that time we presented a block diagram of the electronics which would be useful to reproduce here:

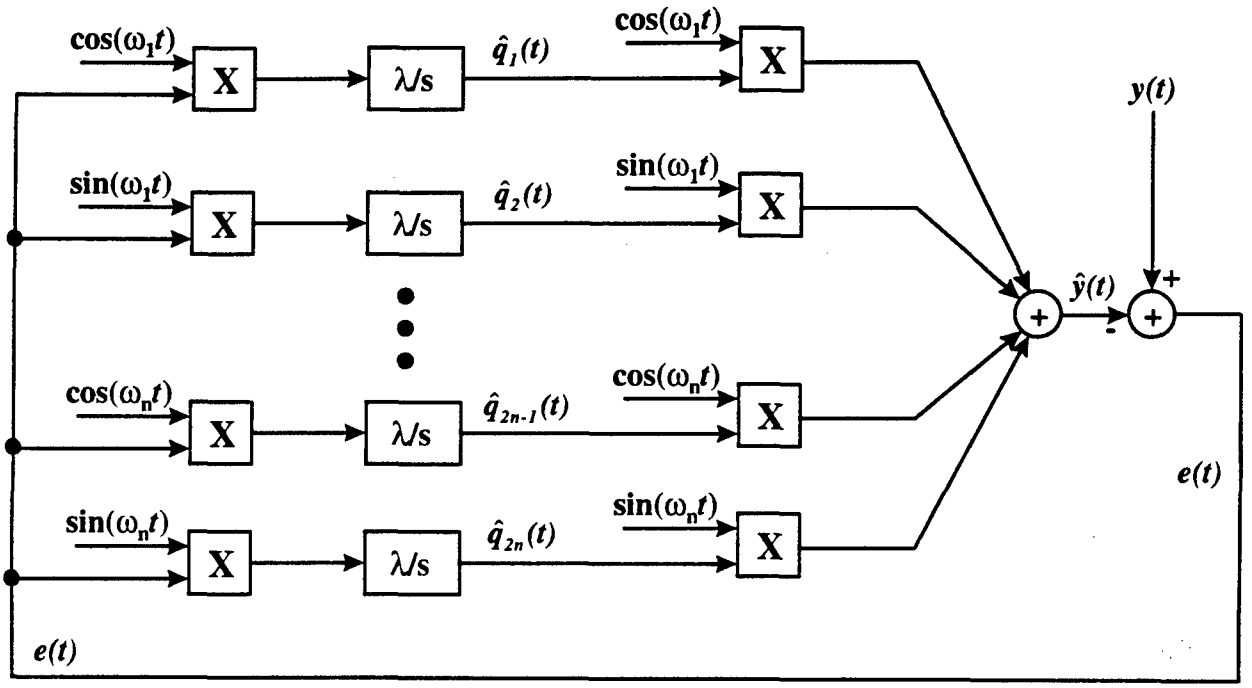


Figure 10-2. Block Diagram of the Multi-Tone Demodulator

The demodulator operates with $n = 3$. In addition, the cosine and sine signals are scaled, as are the multiply operations. We will not elaborate on the effects this has on system behavior, except to say that they are simple. As long as the total loop gain is maintained, the only effect is on the scaling of the weight estimates.

We now describe how each of the above functions is implemented in the analog hardware. Refer to Appendix B for the schematics, which are on four sheets.

Before proceeding, it is essential to note that the multiplier is implemented with an Analog Devices AD734 chip, which has the behavior:

$$W = \frac{(X1 - X2)(Y1 - Y2)}{10} + Z2$$

where all signals are in volts and the W output signal is attached to $Z1$ for feedback. Sheets 1, 2, and 3 of the schematic are nearly identical, and implement the three pairs of operations of the demodulator for the three distinct tones, except for the summations. For example, on Sheet 1, chip U1 performs the cosine multiply, U3 the sine. Chip AR1 pins 5, 6, and 7 implement the integrator for the cosine signal, and AR1 pins 8, 9, and 10 implement the integrator for the sine signal. AR1 pins 1, 2, and 3 implement an amplifier to scale the $\tilde{q}_1(t)$ weight estimate, labeled as EC1 on the schematic. The final multiply prior to summing is performed by the chip U2 for the cosine signal, U4 for the sine signal. The sine and cosine contributions are summed in U2 prior to being sent to the summing circuit on Sheet 4.

The two summation operations are performed on Sheet 4, chip AR4, pins 1, 2, and 3. The sensor signal passes through two filtering sections prior to entering the summing junction. These are twin-T resonant band-pass filters with a peak gain of 3, a damping of 0.25, and center frequencies of 5 and 20 Hz, respectively. They produce a band pass effect between 5 and 20 Hz, with 40 dB/decade skirts. At the summing op amp, the gain from W5 to pin 1 of AR4 is 2, while the gain from any of the Y signals originating from Sheets 1-3 is 1. This relationship is due to the resistor values chosen in this inverting summer. The output from pin 1 of AR4 is the error signal E(T) on the schematic, $e(t)$ of Figure 6-2.

The remainder of Sheet 4 of the schematic implements the modulation of the signals emanating from the D/A converters. These enter as UC1 and US1 as the real and imaginary components of the command for the first tone, UC2 and US2 for the second tone, etc. The output of the sine multiply is added in as signal Z2 to the result of the cosine multiply (e.g. to U19). The result is fed to the inverting, gain of 1 summer of AR4, pins 8, 9, and 10. A final inverting gain change is used before the signal is output to the appropriate actuator (determined by soldering a jumper into one of W2, W3, W4, W7, W9, or W10).

U19 is a DC/DC converter used to stabilize supply voltages on the card.

10.5 Synchronization Signal Cards

The synchronization signal card has two functions:

1. Convert a pulse train into a cosine and a quadrature sine signal of the same frequency as the input train.
2. Amplify and filter the sensor signals to minimize noise to the analog demodulators. Six channels of input are available, although only 3 are used in the ANC system.

The schematic for this card is found in Appendix B and consists of 9 sheets. The first 4 sheets implement the first function. Sheets 5 and 6 implement the second function. Part of Sheet 6 and Sheets 7 and 8 implement processing for other sensors and are not used. Sheet 9 shows the DC/DC converter used to stabilize the supply voltages on the card.

The second function of the card, residing on Sheets 5 and 6 is discussed first because it is much simpler. The circuits of interest are all identical, so only the first will be discussed, encompassing the circuits of AR7 pins 1, 2, and 3, and AR7 pins 5, 6, and 7.

The input circuit was designed to accommodate the input from a PCB type accelerometer, which modulated its sensor signal on the same wires as the supply voltage, so that only two conductors are needed. Since that is not the function of this circuit, the diode part of the circuit was cut out, beginning above C194. The remaining circuit is a one-pole, inverting, high-pass filter with a corner at 1.6 Hz and a gain of 4.7 (with the value of R38 at 4.7K). The following stage, AR7 pins 5-7, is just an inverting low pass filter with a gain of 10 and a corner frequency of approx. 30 Hz (with C12 at 0.27 μF). The output of this circuit is then passed to the demodulator for the first sensor. The other circuits on this Sheet and the top of the next operate in an analogous fashion.

The synch to cos/sin converter circuit schematics are shown on Sheets 1 through 4. It has a block diagram as shown in Figure 10-3 below.

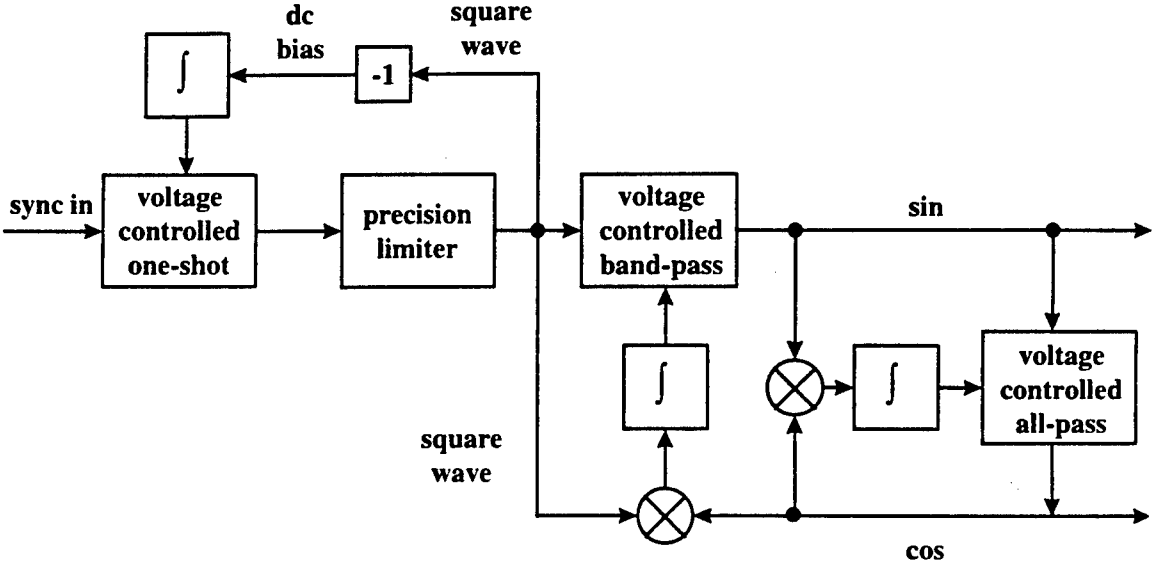


Figure 10-3. Block Diagram of the Synch to Cos/Sin Converter Circuit

The circuit works by creating a well formed square wave at the input frequency and filtering out its higher harmonics to produce a sine wave. The cosine wave is produced by shifting the phase of the sine wave by 90 degrees using a first order all pass. All frequency dependencies of the circuit are controlled to maintain phase shifts and amplitude peaks.

The synchronization pulse train is assumed to be of TTL level but has few other known attributes. The first task is to create a square wave of constant amplitude and of exactly 50% duty cycle. This is done by using the sync signal to trigger a one-shot pulse generator, the duration of which is controlled by a voltage. The control voltage is developed by integrating the square wave output. Any deviations from 50% duty cycle create a DC bias in the average value of the wave. The precision limiter is used to ensure that the square wave is always switching between ±5 volts. Referring to Sheet 1, the variable one shot consists of the open collector

output, Analog Devices LM139 comparator U9, pins 1, 6, and 7, the comparator U10, and U9, pins 2, 4, and 5 and surrounding circuitry. The bias integrator is AR1, pins 12, 13, and 14. The precision limiter is constructed from the analog switches of U3.

The next stage of the circuit is the voltage-controlled band-pass filter, the output of which is the sine wave, on pin 14 of AR2. This is essentially a twin-T two-pole resonant band-pass, the frequency of which is controlled with what amounts to a voltage-controlled resistor formed by the analog multiplier U12. The control voltage emanates from the integrator output, pin 7 of AR2. The square wave and the cosine are nominally in quadrature, so that their product should produce no DC bias. However, phase shift errors in the sine wave resulting from improper centering of the band-pass filter frequency response must produce similar shifts in the cosine wave, resulting in a DC bias for the product of the cosine and the square waves. This product is created by U13 at pin 12, integrated, and used to correct the center frequency of the band-pass.

Finally, on Sheet 4, the sine wave is passed through a one-pole all pass filter $\frac{(s-\omega_o)}{(s+\omega_o)}$ to produce the cosine wave. This filter has a gain of 1 over all frequency, but its phase varies from -180 degrees at DC to 0 degrees at infinity. It attains -90 degrees at the frequency $s = j\omega_o$. This is the desired situation. The center frequency is adjusted with another voltage controlled resistor consisting of U25, with an input coming from the integrator output at pin 1 of AR2. The integrator has as its input the product of the sine and cosine waves, which has zero DC bias if the phase between the sine and cosine is maintained at 90 degrees. Variations in the phase cause a DC bias which integrates to shift the center frequency of the all-pass until the phase is corrected.

Two more circuits identical with the one just described are implemented for the remaining two tones on Sheets 2, 3, and 4.

This clever circuit was designed by Mr. Bill Burton, a principal electrical engineer at Harris GASD.

10.6 Custom Connector Wire List

The custom D-connectors that connect the analog demodulator cards with the A/D and D/A converters have a wire list as shown below in Table 10-1. The signals are grouped by demodulator card. Each demodulator card has one demodulator for three tones, corresponding to one sensor, and one modulator for three tones, corresponding to one actuator. The signals are grouped by demodulator. The terminology "y cos 1" indicates that this is the cosine component of tone 1 for this demodulator card's sensor measurement. Similarly, "u sin 2" indicates that this is the sine component of tone 2 for this demodulator card's actuator command.

The pin number on the demodulator card's D connector is the second column. The destination card for the wire is listed in column 3. The label on the corresponding connector is in column 4. The pin number on the connector is in column 5, and D/A channel or A/D channel of interest is listed in the remaining two columns.

Table 10-1. Wire List for the Custom Connector

Signal	Demod Pin	Dest. Card	Connect. No.	Connect. Pin	D/A Channel	A/D Channel
Demod 1						
y cos 1	1	Mux 1	P3	24		0
y sin 1	2	Mux 1	P3	10		1
y cos 2	3	Mux 1	P3	21		2
y sin 2	4	Mux 1	P3	7		3
y cos 3	5	Mux 1	P3	18		4
y sin 3	6	Mux 1	P3	4		5
u cos 1	7	D/A 1	P9	24	0	
u sin 1	8	D/A 1	P9	10	1	
u cos 2	9	D/A 1	P9	21	2	
u sin 2	10	D/A 1	P9	7	3	
u cos 3	11	D/A 1	P9	18	4	
u sin 3	12	D/A 1	P9	4	5	
a. return	13					
Demod 2	**Demod Pin**	**Dest. Card**	**Connect. No.**	**Connect. Pin**	**D/A Channel**	**A/D Channel**
y cos 1	1	Mux 1	P3	15		6
y sin 1	2	Mux 1	P3	1		7
y cos 2	3	Mux 1	P2	24		8

y sin 2	4	Mux 1	P2	10		9
y cos 3	5	Mux 1	P2	21		10
y sin 3	6	Mux 1	P2	7		11
u cos 1	7	D/A 1	P9	15	6	
u sin 1	8	D/A 1	P9	1	7	
u cos 2	9	D/A 1	P9	23	9	
u sin 2	10	D/A 1	P9	9	10	
u cos 3	11	D/A 1	P9	20	11	
u sin 3	12	D/A 1	P9	6	12	
a. return	13					
Demod 3	**Demod Pin**	**Dest. Card**	**Connect. No.**	**Connect. Pin**	**D/A Channel**	**A/D Channel**
y cos 1	1	Mux 1	P2	18		12
y sin 1	2	Mux 1	P2	4		13
y cos 2	3	Mux 1	P2	15		14
y sin 2	4	Mux 1	P2	1		15
y cos 3	5	Mux 1	P3	12		16
y sin 3	6	Mux 1	P3	23		17
u cos 1	7	D/A 1	P9	17	13	
u sin 1	8	D/A 1	P9	3	14	
u cos 2	9	D/A 1	P9	14	15	
u sin 2	10	D/A 2	P10	24	0	
u cos 3	11	D/A 2	P10	10	1	
u sin 3	12	D/A 2	P10	21	2	
a. return	13					
Demod 4	**Demod Pin**	**Dest. Card**	**Connect. No.**	**Connect. Pin**	**D/A Channel**	**A/D Channel**
y cos 1	1	Mux 1	P3	9		18
y sin 1	2	Mux 1	P3	20		19
y cos 2	3	Mux 1	P3	6		20
y sin 2	4	Mux 1	P3	17		21

y cos 3	5	Mux 1	P3	3		22
y sin 3	6	Mux 1	P3	14		23
u cos 1	7	D/A 2	P10	7	3	
u sin 1	8	D/A 2	P10	18	4	
u cos 2	9	D/A 2	P10	4	5	
u sin 2	10	D/A 2	P10	15	6	
u cos 3	11	D/A 2	P10	1	7	
u sin 3	12	D/A 2	P10	23	9	
a. return	13					
Demod 5	**Demod Pin**	**Dest. Card**	**Connect. No.**	**Connect. Pin**	**D/A Channel**	**A/D Channel**
y cos 1	1	Mux 1	P3	12		24
y sin 1	2	Mux 1	P3	23		25
y cos 2	3	Mux 1	P3	9		26
y sin 2	4	Mux 1	P3	20		27
y cos 3	5	Mux 1	P3	6		28
y sin 3	6	Mux 1	P3	17		29
u cos 1	7	D/A 2	P10	9	10	
u sin 1	8	D/A 2	P10	20	11	
u cos 2	9	D/A 2	P10	6	12	
u sin 2	10	D/A 2	P10	17	13	
u cos 3	11	D/A 2	P10	3	14	
u sin 3	12	D/A 2	P10	14	15	
a. return	13					

Demod 6	Demod Pin	Dest. Card	Connect. No.	Connect. Pin	D/A Channel	A/D Channel
y cos 1	1	Mux 1	P3	3		30
y sin 1	2	Mux 1	P3	14		31
y cos 2	3	Mux 2	P4	24		32
y sin 2	4	Mux 2	P4	10		33
y cos 3	5	Mux 2	P4	21		34
y sin 3	6	Mux 2	P4	7		35
u cos 1	7	D/A 3	P11	24	0	
u sin 1	8	D/A 3	P11	10	1	
u cos 2	9	D/A 3	P11	21	2	
u sin 2	10	D/A 3	P11	7	3	
u cos 3	11	D/A 3	P11	18	4	
u sin 3	12	D/A 3	P11	4	5	
a. return	13					

10.7 Bias Adjustment Procedures

It is quite likely that the integrator biases will need to be adjusted out on the demodulator cards from time to time. In addition, the bias table used by the software may need to be updated occasionally, certainly every time the bias adjustment potentiometers are zeroed, or when the bias table measurements of Subsection 10.8.2 show biases greater than about 50 mV. The following gives the two procedures to use in these cases.

10.7.1 Adjusting Out the Integrator Bias on the Demodulator Cards

This procedure should be followed when the biases measured in the procedure of 10.8.2 get larger than about 50 mV.

1. Turn the ANC system power on. Ensure valid sync signals are being input for all three tones. Ensure that **sensor signals** are **disconnected** or **turned off**. Ensure that the **actuators** are **disabled** or **turned off**. Wait 5 minutes for the electronics to stabilize. Ensure all the switches on the demodulator cards (see Figure 10-4 below) are in the "up" position, as they should be for operation.

2. Run the program "chkdaad" by typing this as a command at the OS-9 prompt. Respond to prompts for files. The first file required is the map for the A/D and D/A converters, "addamap.dat". The second file is the data file to use for commanding the D/A converters, "chkdaad.dat". The program prints out the measured A/D readings every 5 seconds. These should be zero, but are normally small values less than 50 mV. The sin and cosine sensor

readings for the various tones are shown with the corresponding channel number displayed by the chkdaad computer program below in Table 10-2.

3. Using a small screwdriver, adjust each of the potentiometers corresponding to channels with large biases to less than 20 mV or so. The potentiometers as viewed from the front of the VME cage are shown below in Figure 10-4. Counterclockwise rotation of the potentiometers makes the biases more positive. Note that the tone 1 sine potentiometer is above the tone 1 cosine potentiometer, in contrast with the order for the other tones shown in Figure 10-4. Hit control-C to exit the chkdaad program when finished.
4. Rerun the bias table computation of Subsection 10.7.2.

Table 10-2. Channel Map for the CHKDAAD Program

Demod 1	**A/D Channel**	**Demod 4**	**A/D Channel**
tone 1 cosine	0	tone 1 cosine	6
tone 1 sine	1	tone 1 sine	7
tone 2 cosine	12	tone 2 cosine	18
tone 2 sine	13	tone 2 sine	19
tone 3 cosine	24	tone 3 cosine	30
tone 3 sine	25	tone 3 sine	31
Demod 2	**A/D Channel**	**Demod 5**	**A/D Channel**
tone 1 cosine	2	tone 1 cosine	8
tone 1 sine	3	tone 1 sine	9
tone 2 cosine	14	tone 2 cosine	20
tone 2 sine	15	tone 2 sine	21
tone 3 cosine	26	tone 3 cosine	32
tone 3 sine	27	tone 3 sine	33
Demod 3	**A/D Channel**	**Demod 6**	**A/D Channel**
tone 1 cosine	4	tone 1 cosine	10
tone 1 sine	5	tone 1 sine	11
tone 2 cosine	16	tone 2 cosine	22
tone 2 sine	17	tone 2 sine	23
tone 3 cosine	28	tone 3 cosine	34
tone 3 sine	29	tone 3 sine	35

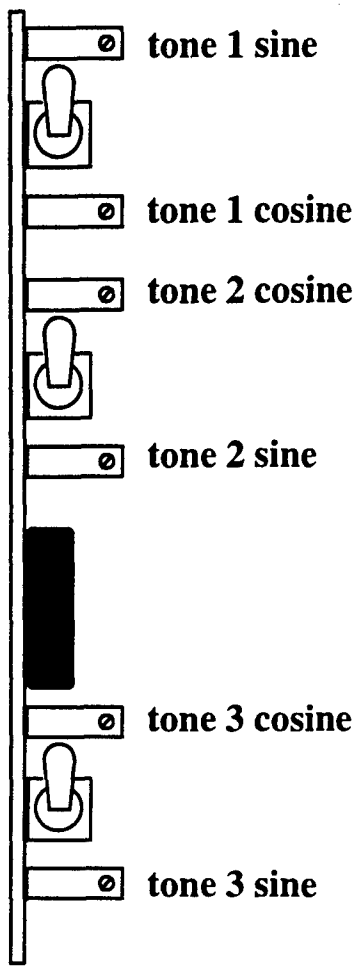


Figure 10-4. Arrangement of the Potentiometers on the Demod. Cards

10.7.2 Recomputing the Bias Table

This procedure should be run at least every time the potentiometers are readjusted on the demodulator cards.

1. Turn the ANC system power on. Ensure valid sync signals are being input for all three tones. Ensure that **sensor signals** are **disconnected** or **turned off**. Ensure that the **actuators** are **disabled** or **turned off**. Wait 5 minutes for the electronics to stabilize. Ensure all the switches on the demodulator cards (see Figure 6-4 above) are in the "up" position, as they should be for operation.
2. Run the program "measbias" by typing this as a command at the OS-9 prompt. Let the program run for about 5 minutes to get good average values for the biases. Hit control-C to exit. The new bias table readings are stored in the file "measbias.dat," where they are read by the "demo" program.

11. Conclusions and Recommendations for Future Work

This report has described efforts undertaken under the Adaptive Neural Control (ANC) Program. Previous developments had theoretically and experimentally demonstrated neural adaptive feedback control and feedforward control for tonal disturbances. The ANC effort rounds out this development by first demonstrating feedforward control for the difficult case of broadband, continuous spectrum disturbances and then integrating and demonstrating a multi-tone, fault-tolerant cancellation in the ASTREX testbed. Thus the demonstrated capabilities cover all disturbance types likely to be of relevance in applications. In addition to the adaptive control of flexible space systems which was the focus of this effort the ANC architecture lends itself to numerous dual-use and commercial applications. In the remainder of this chapter we describe some of these dual-use applications and make recommendations for broader efforts to advance existing ANC capabilities to support these applications.

A key feature of the ANC architecture is its hierarchy of standardized components. This modularity permits great implementation flexibility and also facilitates a multiplicity of diverse applications. As Figure 11-1 illustrates, only a relatively small number of basic components need to be developed and perfected and then combined to support an enormous number of applications. This feature greatly reduces nonrecurring engineering costs and broadens the scope of dual-use activities.

In the area of noise and vibration control, potential applications of ANC range from vibration isolation of precision space sensors to consumer products. In the body of the report, we have already alluded to the use of ANC for autonomous adaptive vibration isolation of space payloads from spacecraft—generated disturbances. There are also many industrial applications including the quieting of machinery in manufacturing facilities and the isolation of semiconductor production facilities from ground vibration.

In addition, we have identified numerous applications for relatively low cost, high volume consumer products. Table 11-1 lists some of the possibilities. As well as noise and vibration applications involving the use of both our commercial actuator technologies and ANC, there are systems that would employ only ANC adaptive controllers. These items include heating and air-conditioning regulators, appliance power regulators and active automobile suspension controls that use a neural controller for fault identification and controller recovery.

Of even broader scope are the applications of ANC outside of noise and vibration control. Some of these are listed in Table 11-2. These items involve ANC in both an adaptive control role or as a passive damage or fault detection system. The latter application would be useful, for example, in the detection and location of damage in civil engineering structures, especially in areas of high seismic activity. Of particular note are two specifically identified applications to medical diagnosis; using passive acoustical measurements to detect failures in Björk Shiley coronary values and to diagnose coronary occlusions. Although, in the former case, some work has been done using a simple multilayer perceptron to give a binary output decision, the general ANC architecture has the potential for greatly improved diagnostic accuracy. Because it can incorporate detailed models of dynamic processes, the ANC architecture offers new dimensions that go well beyond "black box" neural classifiers. These new capabilities could include the ability to fold in directionality and acoustic propagation dynamics information using multiple

sensors, the incorporation of dynamic models of valve operation and mechanical vibration and the use of these internal models to filter out variabilities in the data that are irrelevant. In other words, an ANC system can accept and then refine internal dynamical models to progressively sharpen its diagnostic accuracy.

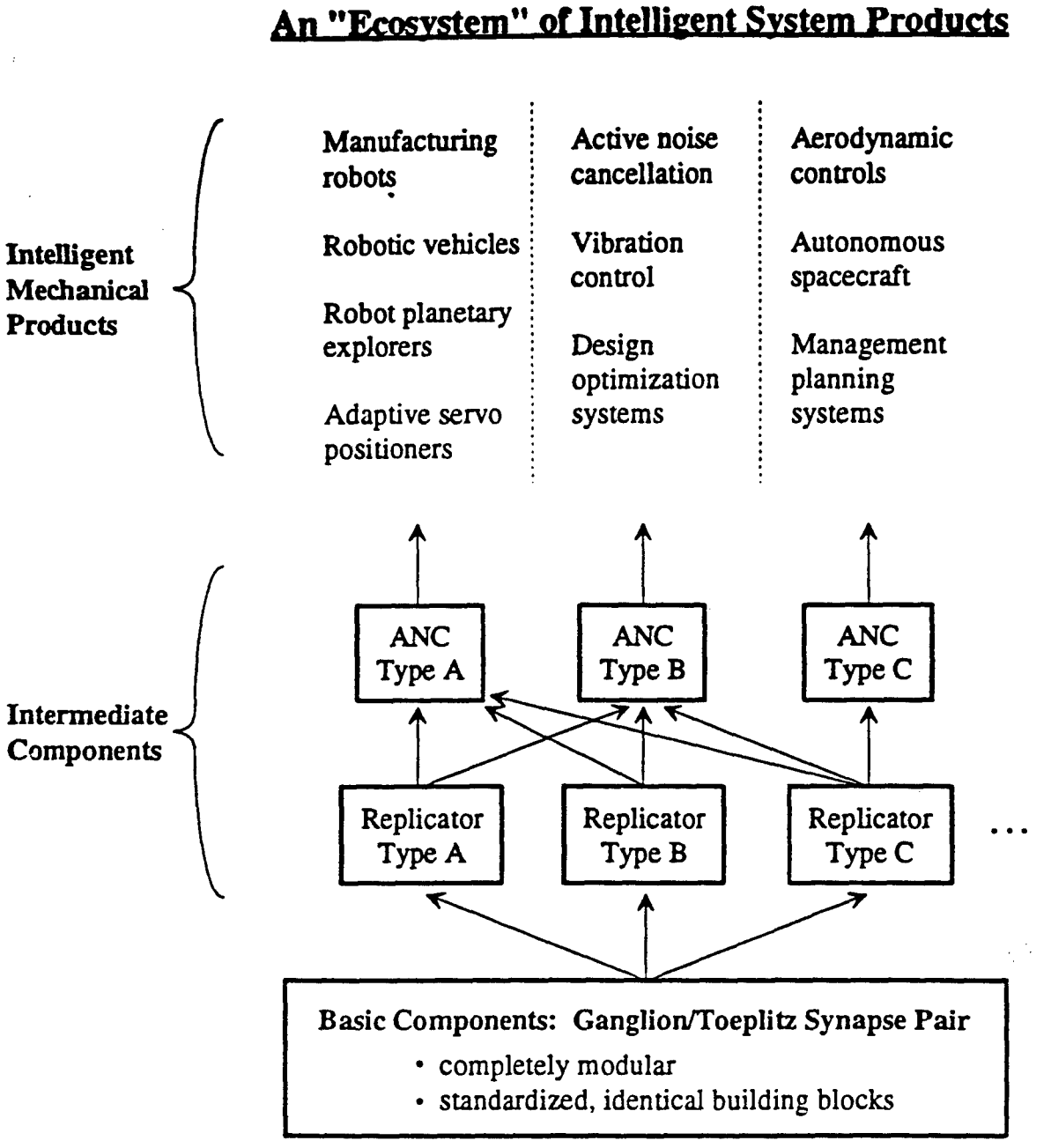


Figure 11-1: Completely Modular, Standardized ANC Components Support a Broad-Based Industry - From Basic Module Supply to Individual Applications

Table 11-1: Potential Consumer Applications of Harris' Vibration Control Hardware and Adaptive Neural Algorithms

Actuator Technology - provides compact, unobtrusive actuation with maximum cost-effectiveness

Neural Adaptive Control - provides "intelligent" control for noise and vibration reduction. Able to work autonomously and adapt to rapidly changing conditions.

Small Actuators + Neural Control

System	Benefits
Washing Machines	less noise, less wear, longer life
Dishwashers	less noise, less wear
Quieter Lawn Mowers, Weed Trimmers, etc.	reduces engine vibration (acoustic transducers and neural controls reduce far field emissions)

Larger Actuators + Neural Control

System	Benefits
Automobile Engine Mounts	eliminates transmission of engine vibration into passenger cabin
Outboard Motors	reduces acoustic noise, hull vibration

Neural Control with Acoustic Transducers

System	Benefits
Vacuum Cleaners	obnoxious noise eliminated for user and bystanders
Hair Dryers	reduces distant noise, reduces interference with other activities (conversation, TV, etc.)
Quieter Lawn Mowers, Weed Trimmers, etc.	reduces far field acoustic emissions, improves user comfort and neighborhood tranquillity

Neural Controls

System	Benefits
Heating/Air-Conditioning	maintains desired temperature while minimizing power consumption
Appliance power regulators (outlet attachment, neural control and appliance transducers regulate power consumption)	reduced electricity bills
Active automobile suspension control (neural control for fault ID and controller recovery)	system fault tolerance and reliability

Table 11-2: Adaptive Neural Control Applications Outside Noise & Vibration Control

System	Benefits
Heating/Air-Conditioning	Maintains desired temperature while minimizing power consumption
Appliance power regulators (outlet attachment, neural control and appliance transducers regulate power consumption)	Reduced electricity bills
Robotic (industrial and aerospace) control	Capable of greater autonomy and more flexible programming
Fault tolerant attitude control systems for spacecraft	More nearly trouble-fee operation for earth-pointing satellites, more nearly autonomous capability for extended-mission spacecraft
Neural Positioner for Precision Motion Control	Ultra precision machining for greater precision and reliability for manufacturing
Neural system for fault ID and control recovery in active automobile suspensions	Neural system does not do the suspension control directly - but oversees the system to detect and react to malfunctions. This secures fault tolerance for active suspension systems - *the* biggest obstacle to OEM acceptance
Damage detection for civil engineering structures (buildings and bridges)	No active control involved. Neural system detects and locates structural weakness or damage using structure-mounted sensors (and issues warnings about the damage detected). Benefits: Forestalls high damage in earthquake prone areas by providing timely diagnosis of "weak spots"
Neural Control for Chemical Processes	Provides autonomous operation. Reduces engineering development and operating costs in the chemical process industry
SLS (Single Leg Separation) detection in Björk-Shiley values	Much more than a "black-box" classifier: * network organizes event groupings and feature classifications * system accepts and then refines internal dynamical models to progressively sharpen its diagnostic accuracy * because it correlates acoustic data with models, the trained system can offer clues on the possible linkages between data features and physical processes
Noninvasive acoustical detection of coronary occlusions	Above remarks also apply - the action of the neural system is much more comprehensive (and potentially much more accurate) then just an "adaptive line enhancer"

Of course, most of the commercial and medical applications sketched above are well into the future. A great deal needs to be done to perfect ANC systems, even for the space or airborne DoD applications.

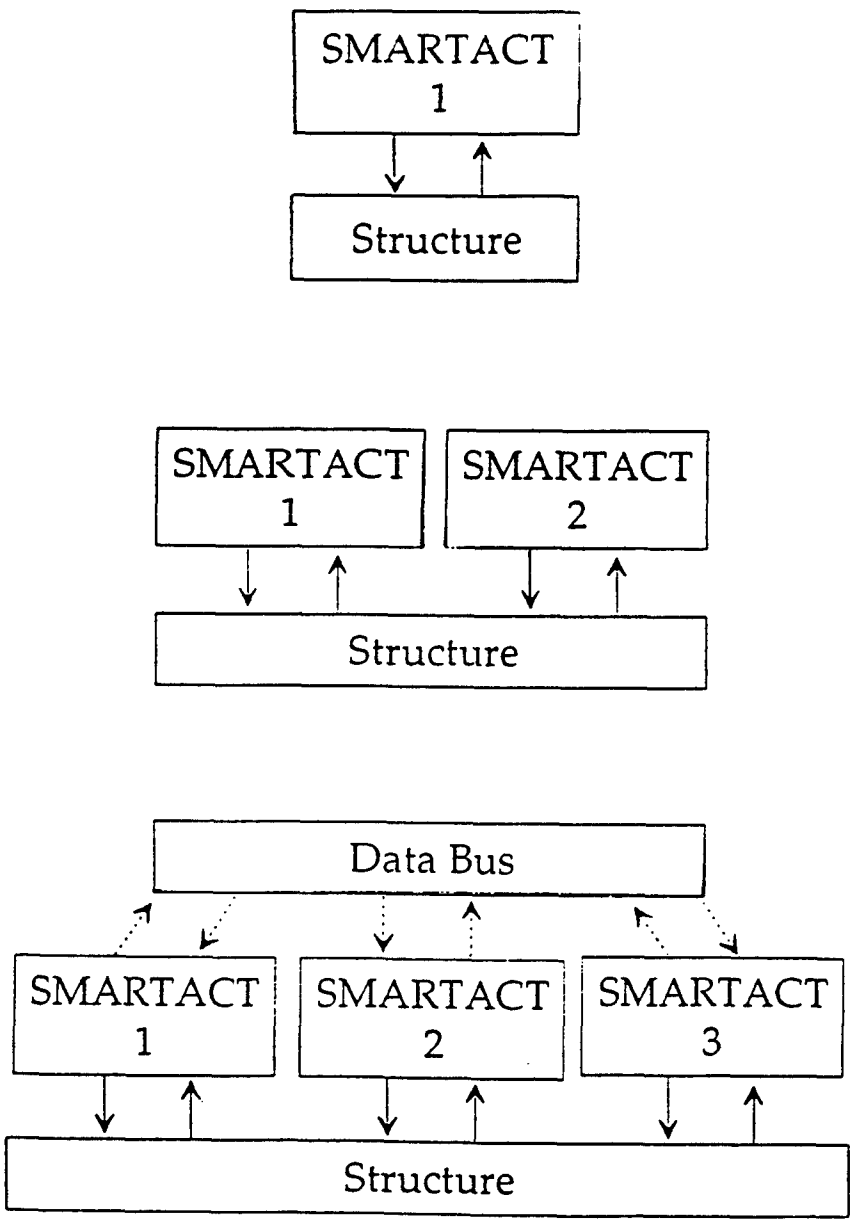
One important area of development is implementation refinement. To date for the most part, we have been using existing DSP hardware. This is appropriate for the early stages of development when it is necessary to accumulate sufficient experience in the operational characteristics of ANC algorithms to be able to formulate meaningful performance and design specifications for improved, dedicated hardware. We are nearly at the point at which such specifications can be elaborated. It is clear that there are two distinct directions that can be taken in hardware improvement. The first is to develop analog implementations. Although the simplest algorithms have been demonstrated with analog hardware, this remains to be done for the more complex systems. Analog implementation eliminates the need for a computer and associated software and reduces cost for many applications. Since the more sophisticated ANC systems will still require digital capability, the second direction for advancement is the development of ASICs for specific applications. Such customized ICs would likely embody a number of standardized, higher-level modules in the ANC architecture. Properly designed, these could serve as the basic building blocks from which a variety of diverse systems could be built.

A second broad area for development deals with the refined *integration* of ANC hardware with actuator hardware and with structural components. Harris has, over the past decade, developed several lines of actuator hardware (both inertial and intrastructural). A natural extension is to integrate the intelligent control capability of ANC directly into each (collocated) actuator/sensor unit. One obtains structures that are "smart" because the actuators, joints and fittings are intelligent. This "smart actuator" concept relies on the fact that even the global, centralized ANC model reference adaptive control algorithm can be re-partitioned into parallel components, each of which resides within one collocated sensor-actuator unit. Figure 11-2 shows the basic concept. The forward and backward path signal flows associated with the particular sensor and actuator unit are handled locally. At the same time, weight adjustment computations associated with interactions among the sensor/actuator units are executed using forward and backward signal information from the other sensor/actuator units transmitted through a data bus. Thus, all computations are carried out within the local processors, yet by virtue of the inter-processor communication, all processors (that are in communication) work in combination to address the multi-input, multi-output centralized control problem.

A further possibility is the integration of ANC technology into smart structures involving embedded "mesoscale" actuation and sensing devices dispersed within the structural material. Clearly, the development of mesoscale actuators realizes the desirable feature of highly redundant and distributed sensory and actuation devices. However, to properly exploit this technology one needs some form of controller that is adaptable, autonomous, highly parallel and fault tolerant. For this reason, the implementation of massively parallel, decentralized adaptive control algorithms (in forms that can be explained in terms of neural network algorithms) seems a perfect complement to mesoscale actuation technology.

Intelligent Actuation

- processing is distributed among the actuator/sensor units
- all the actuator/sensor units that are in communication with one another work together to solve (and implement) the optimal centralized control law



One unit: converges to the optimal SISO control law

Two or more noncommunicating units converge to the optimal decentralized control (with local SISO loops)

Two or more units communicating through a data bus converge to the optimal centralized, MIMO control

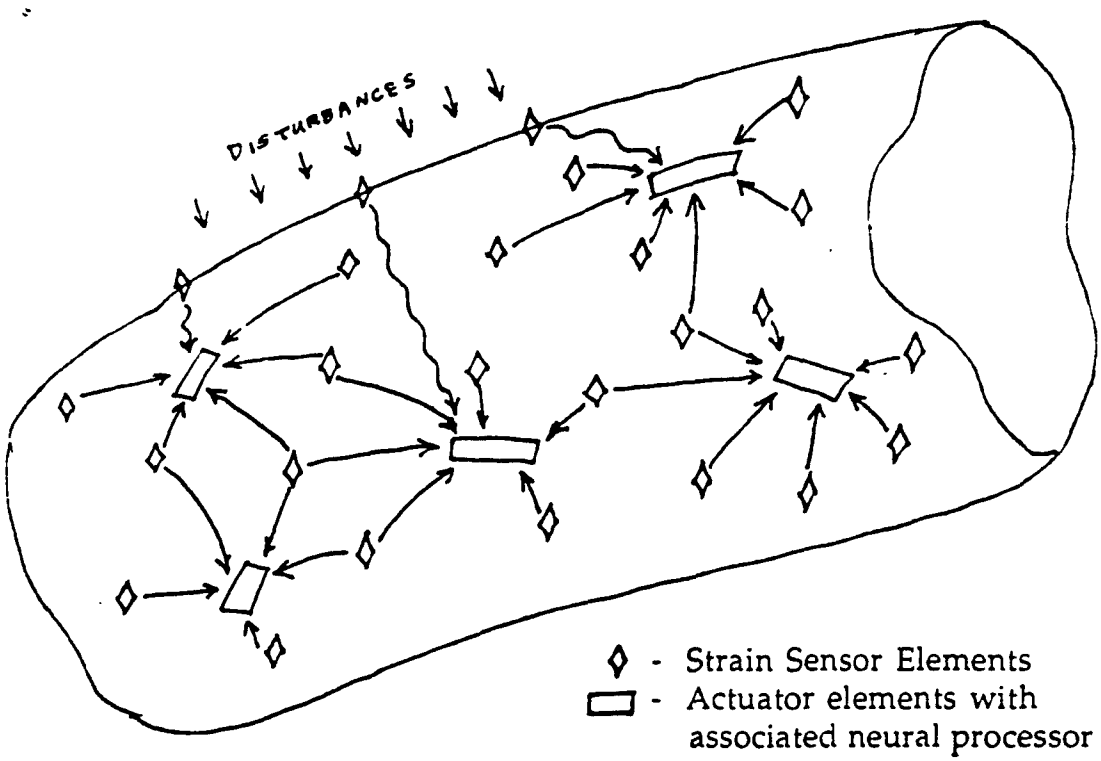
But, as in all previous cases, all processing still occurs within the individual units

Figure 11-2: SMARTACT's (Smart Actuators)

The ANC systems considered to-date all involve a *centralized* information pattern—i.e. full connectivity among all the sensors and actuators. However, in systems having a large number of small actuation and sensing devices spatially distributed throughout the structural matrix, centralized control entails an impractically large number of interconnections. Thus, for application of adaptive neural control to mesoscale actuation systems some form of decentralization constraint needs to be imposed on the controller structure. There are a variety of ANC structures for implementing this kind of decentralized control, with suitable limited interconnection paths, that is needed for coordination of a large number of sensors and actuators.

The most straightforward approach involves the basic ANC controllers described above, but with interconnection constraints. What this means is that some of the synaptic connections that would otherwise exist in a centralized system are removed, leaving only a subset. The remaining synaptic connections correspond, one-to-one, to a decentralized information pattern, e.g. each actuator being driven by a subset of sensors that are spatially proximate to the actuator.

This pattern of interconnections is illustrated by Figure 11-3. The decentralized learning inherent in ANC readily accommodates such decentralization in the control-indeed none of the fundamental stability and convergence properties are modified. Decentralized information patterns simply mean that only a subset of synaptic connections are active and with respect to the corresponding weights, the system is still a gradient descent machine. A decentralized ANC controller still seeks a least mean-square error condition, but operates under the constraints imposed by decentralized structure. When a decentralized control structure is imposed, then the controller can obviously be decomposed into independent sub-controllers-for example, one such subcontroller would activate one actuator, coordinating its set of geographically proximate sensors. This would permit the use of small, computationally simple processing units that are spatially distributed throughout the structure.



- Embedded sensors and actuators widely dispersed within the structural material
- Co-location not necessary
- Parallel neural processors - each one associated with one actuator - each processor very simple (few neurons)
- Each actuator processor connected with a spatially local subset of sensors
- Without detailed model information, processors implement adaptive vibration damping and disturbance cancellation

Figure 11-3: Distributed Adaptive Neural Architecture for Intelligent Structures

12. References

1. D.C. Hyland, J. A. King and D.J. Phillips, "Streamlined Design and Self Reliant Hardware for Active Control of Precision Space Structures", NASA Contractor Report 4637, December 1994.
2. K.S. Narendra, E.G. Kraft and L.H. Ungar, "Neural Networks in Control Systems," *ACC Workshop Notes, 1991 American Control Conference*, Boston Massachusetts, 1991.
3. D.C. Hyland, "Neural Network Architecture for On-Line System Identification and Adaptively Optimized Control," *Proc. IEEE Conf. Dec. Contr.*, Brighton, U.K. December 1991.
4. D.C. Hyland and J.A. King, "Neural Network Architectures for Stable Adaptive Control, Rapid Fault Detection and Control System Recovery," 15th *Annual AAS Guid. Contr. Conf.*, Keystone, CO, February 1992.
5. D.C. Hyland, "Adaptive Neural Control (ANC) Architectures: a Tutorial," *Proceedings of the Industry, Government and University Forum on Adaptive Neural Control for Aerospace Structural Systems*, August 19-20, 1993 Melbourne, FL.
6. J.-N Juang, M. Phan, L.G. Horta and R.W. Longman, "Identification of Observer/Kalman Filter Markov Parameters: Theory and Experiments," *Proceedings of the AIAA Guidance, Navigation and Control Conference*, New Orleans, Louisiana, August 1991.
7. M. Phan, "New Concepts of Adaptive Neural Control," *Proceedings of the Industry/Government Forum on Adaptive Neural Control for Aerospace Structural Systems*, August 19-20, 1993, Melbourne, FL.
8. A.H. Jazwinski, *Stochastic Processes and Filtering Theory*, Section 7.3, pp. 200ff, Academic Press, 1970.
9. C. L. Thornton, G. J. Bierman, *Filtering and Error Analysis via the UDU^T Factorization*, IEEE TAC, Vol. AC-23, No. 5, pp. 901-907, October, 1978.

Appendix A. C Code

Main Routine (demo.c)

```
/*=================================================================*/
/* Main ANC control task. Kalman filter algorithm with state save. */
/*=================================================================*/

#include   <stdio.h>
#include   <math.h>
#include   "ad_lib.h"
#include    "da_lib.h"
#include    "floatutl.h"
#include   "cmplx6.h"
#include   "kalman2.h"

/*====================== IMPORTANT CONSTANTS ====================*/

#define CHANS (38)
#define SAMPLE_PERIOD (20.0)          /* seconds */
#define ULIMIT (5.0)                  /* volts maximum allowed out */
#define NU (6)                        /* number of control outputs */
#define NY (3)                        /* number of measurements    */
#define MAXN (6)                      /* max NU or NY */

char *filenames[] = {"all_ok.dat",
                     "fail_3.dat",
                     "fail_3_6.dat"};

/*========================== MACROS ==============================*/

#define trunc(v) (1.e-3 * floor((v)*1.e3))

#define SORT(dst,src,vec,cnt) \
{ for (i = 0; i < cnt; i++) \
  { dst[i] = src[vec[i]]; \
  } \
}
#define UNSORT(dst,src,vec,cnt) \
{ for (i = 0; i < cnt; i++) \
  { dst[vec[i]] = src[i]; \
  } \
}

#define PRINT_STEP_HEADER \
printf("\n====== New Step, actuators %d to %d, sensors %d to %d
======", \
     0, (NU-1), 0, (NY-1));

#define PRINT_Y_U \
{ printf("\n---------------------- controller %d: ------------------
-",k); \
  printf("\ny: "); \
  for (i = 0, l = 0, j = 2*k*MAXN; i < NY; i++, j+=2) \
  { if (l < 4) { l++; } else { l = 0; printf("\n   "); } \
    printf(" (%7.3f,%7.3f)",trunc(yy[j]),trunc(yy[j+1])); \
```

```
  } \
  printf("\nu: "); \
  for (i = 0, l = 0, j = 2*k*MAXN; i < NU; i++, j+=2) \
  { if (l < 3) { l++; } else { l = 0; printf("\n    "); } \
    printf(" (%7.3f,%7.3f)",trunc(uu[j]),trunc(uu[j+1])); \
  } \
  if (controllers[k]->ipert < NU) \
  { printf("\nperturbation of %f at control input %d", \
      controllers[k]->r,controllers[k]->ipert); \
  } \
  else if (controllers[k]->ipert < 2*NU) \
  { printf("\nperturbation of %f at control input %d", \
      -controllers[k]->r,controllers[k]->ipert-NU); \
  } \
  else \
  { printf("\nno perturbation"); \
  } \
}

/*====================== Control C Handler ==================*/

int ctlC_flag = 0;

int  handle_ctlC(signal)
int signal;
{ ctlC_flag = (signal < 32);
}

/*========================== MAIN ==========================*/

main()
{
    int i, j, k, l, sleep_count, ok;
    FILE *fp;
    char ans[5];

    int  advec[CHANS], davec[CHANS];
     float adbuff[CHANS], dabuff[CHANS], uu[CHANS], yy[CHANS];
     float yybias[CHANS];

     controller controllers[3];

    /* Convert from seconds to sleep count */

    sleep_count = (int) (SAMPLE_PERIOD * 100.0);

    /*****************************************************************/
    /* initialization of da and ad mapping vectors advec, davec */
    /*****************************************************************/

    fp = fopen("addamap.dat","r");

    for (i = 0; i < CHANS; i++)
    { fscanf(fp,"%d",&advec[i]);
    }

    for (i = 0; i < CHANS; i++)
    { fscanf(fp,"%d",&davec[i]);
```

```
    dabuff[i] = 0.0;  uu[i] = 0.0;
  }

  fclose(fp);

  fp = fopen("measbias.dat","r");

  for (i = 0; i < CHANS; i++)
  { fscanf(fp,"%e",&yybias[i]);
  }

  fclose(fp);

  /*****************************************/
  /*    A/D and D/A Initialization         */
  /*****************************************/

  ad_setup(0,CHANS);
  da0_setup();
  da1_setup();
  da2_setup();

  /*****************************************/
  /*     Controller Initialization         */
  /*****************************************/

  controllers[0] = control_new(NY,NU,ULIMIT);
  controllers[1] = control_new(NY,NU,ULIMIT);
  controllers[2] = control_new(NY,NU,ULIMIT);

  /* Load data for restart. */

  do
  { printf("\n");
    printf("\nRestart Option (1 to 3):");
    printf("\n1 -- All Actuators Working");
    printf("\n2 -- Actuator 3 Failed");
    printf("\n3 -- Actuators 3 & 6 Failed");
    printf("\n4 -- Start Learning from Scratch");
    printf("\n5 -- Exit \n> ");

    gets(ans,4);

  } while (ans[0] < '1' || ans[0] > '5');

  if (ans[0] >= '1' && ans[0] <= '3')
  { fp = fopen(filenames[*ans-'1'],"r");

    control_restart(fp,controllers[0]);
    control_restart(fp,controllers[1]);
    control_restart(fp,controllers[2]);

    fclose(fp);

    printf("\n\nRestarting from stored data in file <%s><\n\n",
           filenames[ans[0] - '1']);
  }
  else if (ans[0] == '5')
```

```
{ printf("\n");
  exit(0);
}

/* install handler for ^C */
 intercept(handle_ctlC);

/*****************************************/
/*    Main control loop                  */
/*****************************************/

 while(!ctlC_flag)
 {
    /* Get A/D converter data */

       ad_convert(adbuff,CHANS);

    /*=======================================================*/
    /*  Sort adbuff into yy.  Apply control algorithm(s).    */
    /*  Unsort uu back into dabuff.                          */
    /*=======================================================*/

       SORT(yy,adbuff,advec,CHANS);

    for (i = 0; i < CHANS; i++)
    { yy[i] -= yybias[i];
    }

    PRINT_STEP_HEADER;

    control_iterate(&yy[0],&uu[0],controllers[0]);
    k = 0; PRINT_Y_U;

    control_iterate(&yy[12],&uu[12],controllers[1]);
    k = 1; PRINT_Y_U;

    control_iterate(&yy[24],&uu[24],controllers[2]);
    k = 2; PRINT_Y_U;

    printf("\n");

    UNSORT(dabuff,uu,davec,CHANS);

    /*********************************/
    /* Output D/A Converter Data     */
    /*********************************/

       da0_convert(dabuff,0,16);
       da1_convert(&dabuff[16],0,16);
       da2_convert(&dabuff[32],0,(CHANS-32));

    /* Wait the requisite time before taking more data. */

    tsleep(sleep_count);
 }

/* Zero out the D/A converters. */
```

```
    for (i = 0; i < CHANS; i++)
      dabuff[i] = 0.0;

     da0_convert(dabuff,0,16);
     da1_convert(&dabuff[16],0,16);
     da2_convert(&dabuff[32],0,(CHANS-32));

     printf("\nda converters have been zeroed! \n\n");

    /* Save data for later restart. */

    fp = fopen("demo.dat","w");
    control_save(fp,controllers[0]);
    control_save(fp,controllers[1]);
    control_save(fp,controllers[2]);
    fclose(fp);
}
```

Kalman Filter Cancellation Algorithm (kalman2.h)

```
/*******************************************************/
/* This file implements the MIMO single tone           */
/* algorithm based on optimal filtering of y.          */
/*******************************************************/

#undef FALSE
#define FALSE (0)

#undef TRUE
#define TRUE (1)

#define TOLER_CTL ((float) 1.0e-8)
#define INITIAL_PERT ((float) 0.1)
#define MIN_PNORM ((float) 0.1)
#define ARM_COUNT(nu) (2*(nu+1))

/*************************************************/
/* controller type definition                    */
/*************************************************/

typedef struct
{ int ny, nu, ipert, arm;
  float lambda, sig2, r, maxu, maxr;
  complex_matrix Phat, Phatinv, Ghat, e, um, ym, eta,
                 Qeta, Qetap, Qetag;
  LDL_decomp Qinv, Qinvold;
} *controller;

/*************************************************/
/* New controller allocation                     */
/*************************************************/
controller control_new(ny,nu,ulimit)
int ny, nu;
float ulimit;
/*************************************************/
{ controller c;

  c = (controller) malloc(sizeof(*c));

  c->nu = nu;
  c->ny = ny;
  c->ipert = 2*nu;
  c->arm = ARM_COUNT(nu);

  c->lambda = 1.0 / ((float) (1 + nu));
  c->sig2 = 0.0;
  c->r = INITIAL_PERT;
  c->maxu = 0.9 * ulimit;
  c->maxr = 0.1 * ulimit;

  c->Phat = cm_new(ny,nu,0.0,0.0);
  c->Phatinv = cm_new(nu,ny,0.0,0.0);
  c->Ghat = cm_new(ny,1,0.0,0.0);
  c->e = cm_new(ny,1,0.0,0.0);
  c->um = cm_new(nu,1,0.0,0.0);
```

```
  c->ym = cm_new(ny,1,0.0,0.0);
  c->eta = cm_new(nu+1,1,0.0,0.0);
  c->Qeta = cm_new(nu+1,1,0.0,0.0);
  c->Qetap = cm_new(nu,1,0.0,0.0);
  c->Qetag = cm_new(1,1,0.0,0.0);

  c->Qinv = LDL_new(nu+1);
  c->Qinvold = LDL_new(nu+1);
  LDL_eye(c->Qinv,TOLER_CTL);

  return c;

} /* control_new */

/*************************************/
/* Controller Deallocation           */
/*************************************/
void control_free(c)
  controller c;
/*************************************/
{ LDL_free(c->Qinvold);
  LDL_free(c->Qinv);

  cm_free(c->Qetag);
  cm_free(c->Qetap);
  cm_free(c->Qeta);
  cm_free(c->eta);
  cm_free(c->ym);
  cm_free(c->um);
  cm_free(c->e);
  cm_free(c->Ghat);
  cm_free(c->Phatinv);
  cm_free(c->Phat);

  free(c);

} /* end control_free */

/*************************************************/
/* Controller restart                            */
/* c is already allocated and zeroed.            */
/* fp is open for read, pointing at ghat         */
/*************************************************/
void control_restart(fp,c)
controller c;
FILE *fp;
/*************************************************/
{ complex_matrix ghat_tmp, phat_tmp, u_tmp, L, D;
  int i, k;

  ghat_tmp = cm_get2(fp);
  phat_tmp = cm_get2(fp);

  if ((ghat_tmp->nrows == c->ny) && (ghat_tmp->ncols == 1))
  { cm_assign(c->Ghat,ghat_tmp);
  }
  else
  { printf("control_restart:  Ghat isn't dimensioned right!");
```

```
  }

  if ((phat_tmp->nrows == c->ny) && (phat_tmp->ncols == c->nu))
  { cm_assign(c->Phat,phat_tmp);
  }
  else
  { printf("control_restart:  Phat isn't dimensioned right!");
  }

  D = cm_get2(fp);
  L = cm_get2(fp);

  for (i = 0; i < c->nu + 1; i++)
  { LDL_D(c->Qinv,i) = cv(D,i)->real;
    for (k = 0; k < i; k++)
    { c_assign(LDL_L(c->Qinv,i,k),cm(L,i,k));
    }
  }

  fscanf(fp,"sig2 %f, r %f\n",&(c->sig2),&(c->r));

  cm_free(D);
  cm_free(L);
  cm_free(ghat_tmp);
  cm_free(phat_tmp);

  /* set states that are not read in */

  c->arm = 4;

} /* control_restart */

/*************************************************/
/* Controller save state for restart             */
/* fp is open for writing.                       */
/*************************************************/
void control_save(fp,c)
controller c;
FILE *fp;
/*************************************************/
{ int i, k, n;
  complex_matrix L, D;

  cm_put(fp,"Ghat",c->Ghat);
  cm_put(fp,"Phat",c->Phat);

  n = (c->Qinv)->n;
  L = cm_new(n,n,0.0,0.0);
  D = cm_new(1,n,0.0,0.0);

  for (i = 0; i < c->nu + 1; i++)
  { cv(D,i)->real = LDL_D(c->Qinv,i);
    for (k = 0; k < i; k++)
    { c_assign(cm(L,i,k),LDL_L(c->Qinv,i,k));
    }
  }

  cm_put(fp,"Qinv diagonal",D);
```

```
  cm_put(fp,"Qinv lower triangular",L);

  fprintf(fp,"sig2 %e, r %e\n",c->sig2,c->r);

  cm_free(L);
  cm_free(D);

} /* control_save */

/*************************************************/
/* control algorithm                             */
/*************************************************/
void control_iterate(y,u,c)
  float *u, *y;
  controller c;
/*************************************************/
{ int i, k, ny, nu, plant_changed;
  float rho, lambda, new_sig2, newr, maxu, certainty_index;

  ny = c->ny;
  nu = c->nu;
  lambda = c->lambda;

  /*****************************************/
  /* Map the measurements onto a complex y  */
  /*****************************************/

  for (i = 0; i < ny; i++)
  { c_assign_real(cv(c->ym,i),y[2*i],y[2*i+1]);
  }

  /*****************************************/
  /* Compute error and uncertainty ratio */
  /* e = ym - Phat*um - Ghat               */
  /* eta = [1; um]                         */
  /* Qeta = inv(Qinv)*eta                  */
  /* rho = real(eta'*Qeta) + 1             */
  /*****************************************/

  cm_mul(c->e,c->Phat,c->um);
  cm_sub(c->e,c->ym,c->e);
  cm_sub(c->e,c->e,c->Ghat);

  c_assign_real(cv(c->eta,0),1.0,0.0);
  cm_assign_part(c->eta,1,0, c->um,0,0, nu,1);

  LDL_inv(c->Qeta,c->Qinv,c->eta);
  rho = cv_dotp(c->eta,c->Qeta) + 1.0;

  /*****************************************/
  /* Check whether the plant has changed. */
  /* Update uncertainty estimates and the */
  /* inverse of Q.                         */
  /*****************************************/

  new_sig2 = cv_norm2(c->e) / (rho * ((float) ny));
  plant_changed = (new_sig2 > 20.0 * c->sig2) && (c->arm == 0);
```

```
if (plant_changed)
{
  /* Qinv = Qsmall + eta*eta' */

  printf("\nplant change detected...");
  LDL_eye(c->Qinvold,TOLER_CTL);
  LDL_update(c->Qinv, 1.0,c->Qinvold, 1.0,c->eta);
  c->arm = ARM_COUNT(c->nu);
  c->r = INITIAL_PERT;
  certainty_index = 0.0;
}
else
{
  /* Qinv = Qinvold + eta*eta' */

  LDL_assign(c->Qinvold,c->Qinv);
  LDL_update(c->Qinv, 1.0,c->Qinvold, 1.0,c->eta);
  if (c->arm > 0) (c->arm)--;
  certainty_index =
        2.0 * sqrt(new_sig2*c->sig2) / (new_sig2 + c->sig2) / rho;
}

certainty_index = MINIMUM(1.0,3.0*certainty_index);
c->sig2 += lambda*(new_sig2 - c->sig2);

/*****************************************/
/* Update plant estimates.               */
/* Qeta = inv(Qinv)*eta                  */
/* Qetag = Qeta(0), Qetap = Qeta(1:nu)   */
/* Ghat += e * Qetag'                    */
/* Phat += e * Qetap'                    */
/*****************************************/

LDL_inv(c->Qeta,c->Qinv,c->eta);
c_assign(cv(c->Qetag,0),cv(c->Qeta,0));
cm_assign_part(c->Qetap,0,0, c->Qeta,1,0, nu,1);

cv_xyH(c->Ghat,c->e,c->Qetag);
cv_xyH(c->Phat,c->e,c->Qetap);

/*********************************************/
/* Adjust perturbation size and direction. */
/* ipert cycles from 0 to 2*nu:            */
/* 0 to nu-1    -- jog u(ipert) positive   */
/* nu to 2nu-1 -- jog u(ipert-nu) negative */
/* 2*nu         -- no perturbation         */
/*********************************************/

newr = cv_norm(c->e) / (MIN_PNORM + cv_norm(c->Phat));
c->r = MINIMUM(newr,10.0*c->r);
c->r = MINIMUM(c->r,c->maxr);

c->ipert = (c->ipert+1) % (2*nu+1);

/*****************************************/
/* Compute final control.                */
/* u = -pseudoinv(Phat)*Ghat + pert      */
/*****************************************/
```

```
  cm_inv2(c->Phatinv,c->Phat);
  cm_mul(c->um,c->Phatinv,c->Ghat);
  cm_scale(c->um,-1.0,c->um);

  maxu = c->maxu;
  cv_limiter(c->um,(c->maxu * certainty_index));

  c->r =  c->r * certainty_index  + (1.0-certainty_index) * c->maxr;

  if (c->ipert < nu)
  { cm(c->um,c->ipert,0)->real += c->r;
  }
  else if (c->ipert < 2*nu)
  { cm(c->um,c->ipert-nu,0)->real -= c->r;
  }

  /*****************************************/
  /* Now assign the complex control to u */
  /*****************************************/

  for (i = 0; i < nu; i++)
  { u[2*i] = cv(c->um,i)->real;
    u[2*i+1] = cv(c->um,i)->imag;
  }

} /* end control_iterate */


/*************************************************/
/* Controller display.                           */
/*************************************************/
void control_display(name, c)
  controller c;
  char *name;
/*************************************************/
{ static char *tf_str[2];
  tf_str[FALSE] = "false";
  tf_str[TRUE] = "true";

  printf("\n\n==> Data for %d meas, %d act controller \"%s\" <==",
         c->ny, c->nu, name);
  printf("\nsig2 = %e, r = %e, ipert = %d\n",c->sig2, c->r, c->ipert);
  cv_display("ym",c->ym);
  cm_display("Phat",c->Phat);
  cv_display("Ghat",c->Ghat);
  LDL_display("Qinv",c->Qinv);
  cv_display("um",c->um);

} /* end control_display) */
```

Complex Matrix Utilities (cmplx6.h)

```
/*******************************************/
/* Simple math stuff                       */
/*******************************************/

#undef C_TOLER
#define C_TOLER (1.0e-8)

#undef MAXIMUM
#define MAXIMUM(a,b) ((a) >= (b) ? (a) : (b))

#undef MINIMUM
#define MINIMUM(a,b) ((a) <= (b) ? (a) : (b))

#undef ABS
#define ABS(a) ((a) >= 0.0 ? (a) : -(a))

#undef LIMITER
#define LIMITER(x,lim) MINIMUM(lim,MAXIMUM(x,-lim))

/*******************************************/
/* Complex type and math.                  */
/*******************************************/

typedef struct
{ float real;
  float imag;
} *complex;

complex c_new(n,Re,Im)
int n;
float Re;
float Im;
{ complex c;
  int i;
  c = (complex) malloc(n*sizeof(*c));

  for (i = 0; i < n; i++)
  {  (c[i]).real = Re;
     (c[i]).imag = Im;
  }

  return c;
}

void c_free(c)
complex c;
{ free(c);
}

void c_assign(to,from)
complex to, from;
{ (to)->real = (from)->real; (to)->imag = (from)->imag;
}

void c_conj(z,x)
```

```
complex z, x;
{ (z)->real = (x)->real;
  (z)->imag = -((x)->imag);
}

void c_assign_real(to,Re,Im)
complex to;
float Re, Im;
{ (to)->real = (Re);
  (to)->imag = (Im);
}

#define c_zero(a) c_assign_real(a,0.0,0.0)

void c_add(c,a,b)
complex c,a,b;
{ (c)->real = (a)->real + (b)->real;
  (c)->imag = (a)->imag + (b)->imag;
}

void c_sub(c,a,b)
complex c, a, b;
{ (c)->real = (a)->real - (b)->real;
  (c)->imag = (a)->imag - (b)->imag;
}

void c_mul(c,a,b)
complex c, a, b;
{ (c)->real = (a)->real * (b)->real  -  (a)->imag * (b)->imag;
  (c)->imag = (a)->real * (b)->imag  +  (a)->imag * (b)->real;
}

#define c_mag2(c) \
((c)->real * (c)->real + (c)->imag * (c)->imag)

#define c_mag(c) \
sqrt(c_mag2(c))

/* Re{x*y'} */
#define c_RexyH(x,y) \
((x)->real * (y)->real + (x)->imag * (y)->imag)

/* z += x * conj(y) */
void c_xyH(z,x,y)
complex z, x, y;
{ (z)->real += (x)->real * (y)->real  +  (x)->imag * (y)->imag;
  (z)->imag += (x)->imag * (y)->real  -  (x)->real * (y)->imag;
}

/* z += x * y */
void c_xy(z,x,y)
complex z, x, y;
{ (z)->real += (x)->real * (y)->real  -  (x)->imag * (y)->imag;
  (z)->imag += (x)->imag * (y)->real  +  (x)->real * (y)->imag;
}

/* z -= x * conj(y) */
void c_mxyH(z,x,y)
```

```
complex z, x, y;
{ (z)->real -= (x)->real * (y)->real  +  (x)->imag * (y)->imag;
  (z)->imag -= (x)->imag * (y)->real  -  (x)->real * (y)->imag;
}

/* z -= x * y */
void c_mxy(z,x,y)
complex z, x, y;
{ (z)->real -= (x)->real * (y)->real  -  (x)->imag * (y)->imag;
  (z)->imag -= (x)->imag * (y)->real  +  (x)->real * (y)->imag;
}

/* z += a * x * y */
void c_axy(z,a,x,y)
complex z, x, y;
float a;
{ (z)->real += (a)*((x)->real * (y)->real  -  (x)->imag * (y)->imag);
  (z)->imag += (a)*((x)->imag * (y)->real  +  (x)->real * (y)->imag);
}

/* z += a * x * yH */
void c_axyH(z,a,x,y)
complex z, x, y;
float a;
{ (z)->real += (a)*((x)->real * (y)->real  +  (x)->imag * (y)->imag);
  (z)->imag += (a)*((x)->imag * (y)->real  -  (x)->real * (y)->imag);
}

/* z += a * x */
void c_ax(z,a,x)
complex z, x;
float a;
{ (z)->real += (a)*((x)->real);
  (z)->imag += (a)*((x)->imag);
}

/* z = a*x */
void c_scale(z,a,x)
complex z, x;
float a;
{ (z)->real = (a)*((x)->real);
  (z)->imag = (a)*((x)->imag);
}

/* prod += x'yz */
void c_xHyz(prod,x,y,z)
complex prod, x, y, z;
{ (prod)->real +=
        (x)->real * ((y)->real*(z)->real - (y)->imag*(z)->imag)
    +   (x)->imag * ((y)->imag*(z)->real + (y)->real*(z)->imag);
  (prod)->imag +=
        (x)->real * ((y)->imag*(z)->real + (y)->real*(z)->imag)
    -   (x)->imag * ((y)->real*(z)->real - (y)->imag*(z)->imag);
}

void c_div(z, x, y)
complex z, x, y;
{ float ymag2;
```

```
  ymag2 = c_mag2(y);
  z->real = (x->real * y->real  +  x->imag * y->imag)/ymag2;
  z->imag = (x->imag * y->real  -  x->real * y->imag)/ymag2;
}

void c_display(title, c)
char *title;
complex c;
{ printf("\n%s  = (%13.6e,%13.6e)\n",title,c->real,c->imag);
}

/*****************************************************************/
/* Complex matrix and vector utilities                           */
/*****************************************************************/

typedef struct
{ int nrows;
  int ncols;
  complex ptr;
} *complex_matrix;

/* Allocate a matrix */

complex_matrix cm_new(n,m,Re,Im)
int n, m;
float Re, Im;
{ complex_matrix z;

  z = (complex_matrix) malloc(sizeof(*z));
  z->nrows = n;
  z->ncols = m;
  z->ptr = c_new(n*m,Re,Im);

  return z;
}

/* Deallocate a matrix */

void cm_free(z)
complex_matrix z;
{ free(z->ptr);
  free(z);
}

/* Get pointer to element (i,k) */
#define cm(z,i,k) \
( ((z)->ptr) + (i) + (k)*((z)->nrows) )

/* Get pointer to element (i) */
#define cv(z,i) \
(((z)->ptr) + (i))

/* z = x */
void cm_assign(z, x)
complex_matrix z, x;
{ int i,k;
  for (k = 0; k < x->ncols; k++)
  { for (i = 0; i < x->nrows; i++)
```

```
    { c_assign(cm(z,i,k),cm(x,i,k));
    }
  }
}

/* z(iz+i,kz+k) = x(ix+i,kx+k) for i = 0:(nr-1), k = 0:(nc-1) */
void cm_assign_part(z,iz,kz, x,ix,kx, nr,nc)
complex_matrix z, x;
int iz, kz, ix, kx, nr, nc;
{ int i,k;
  for (k = 0; k < nc; k++)
  { for (i = 0; i < nr; i++)
    { c_assign(cm(z,iz+i,kz+k),cm(x,ix+i,kx+k));
    }
  }
}

/* z(:,k) = v(:,i) */
cm_assign_col(z,k,v,i)
complex_matrix z, v;
int k,i;
{ int r;
  for (r = 0; r < z->nrows; r++)
  { c_assign(cm(z,r,k),cm(v,r,i));
  }
}

/* new z = x */
complex_matrix cm_copy(x)
complex_matrix x;
{ complex_matrix z;
  z = cm_new(x->nrows,x->ncols,0.0,0.0);
  cm_assign(z,x);
  return z;
}

/* new z = x(ib:ie,kb:ke) */
complex_matrix cm_copy_part(x,ib,ie,kb,ke)
complex_matrix x;
int ib, ie, kb, ke;
{ complex_matrix z;
  int nrow, ncol;
  nrow = (ie-ib+1);
  ncol = (ke-kb+1);
  z = cm_new(nrow,ncol,0.0,0.0);
  cm_assign_part(z,0,0,x,ib,kb,nrow,ncol);
  return z;
}

/* z = 0 */
void cm_zero(z)
complex_matrix z;
{ int i,k;
  for (k = 0; k < z->ncols; k++)
  { for (i = 0; i < z->nrows; i++)
    { c_assign_real(cm(z,i,k),0.0,0.0);
    }
  }
```

```
}

/* z = I */
void cm_eye(z)
complex_matrix z;
{ int i, n;
  cm_zero(z);
  n = MINIMUM(z->nrows,z->ncols);

  for (i = 0; i < n; i++)
  { c_assign_real(cm(z,i,i),1.0,0.0);
  }
}

complex_matrix cm_get(fp)
FILE *fp;
{ int nr, nc, i, k, readok;
  complex_matrix a;

  fscanf(fp,"%d%d",&nr,&nc);

  if ((nr < 1) || (nc < 1))
  { printf("\ncm_get:  nrows or ncols < 1!");
    return ((complex_matrix) NULL);
  }

  a = cm_new(nr,nc,0.0,0.0);

  for (i = 0; i < a->nrows; i++)
  { for (k = 0; k < a->ncols; k++)
    { readok = fscanf(fp,"%e%e",&(cm(a,i,k)->real),&(cm(a,i,k)-
>imag));

      if (readok == EOF)
      { printf("cm_get:  error reading element (%d,%d)!",i,k);
        return ((complex_matrix) NULL);
      }
    }
  }

  return a;
}

void fskipline(fp)
FILE *fp;
{ char c;
  c = getc(fp);
  while (c != '\n')
  { c = getc(fp);
  }
}

complex_matrix cm_get2(fp)
FILE *fp;
{ int nr, nc, i, k, readok;
  complex_matrix a;
  char line[80];
```

```
   fskipline(fp);

   fscanf(fp,"%d by %d",&nr,&nc);

   fskipline(fp);

   if ((nr < 1) || (nc < 1))
   { printf("\ncm_get:  nrows or ncols < 1!");
     return ((complex_matrix) NULL);
   }

   a = cm_new(nr,nc,0.0,0.0);

   for (i = 0; i < a->nrows; i++)
   { fskipline(fp);
     for (k = 0; k < a->ncols; k++)
     { readok = fscanf(fp,"%e %e",&(cm(a,i,k)->real),&(cm(a,i,k)-
>imag));
       if (readok != 2)
       { printf("cm_get:  error reading element (%d,%d)!",i,k);
         cm_free(a);
         return ((complex_matrix) NULL);
       }
     }
     fskipline(fp);
   }

  return a;
}

void cm_put(fp,title,a)
FILE *fp;
char *title;
complex_matrix a;
{ int i, k, l;

  fprintf(fp,"Complex Matrix %s\n %d by %d",title, a->nrows, a-
>ncols);

  for (i = 0; i < a->nrows; i++)
  {
    fprintf(fp,"\nROW %d\n",i);

    for (k = 0, l = 0; k < a->ncols; k++, l++)
    {
      if (l == 2)
      {  l = 0;
         fprintf(fp,"\n");
      }

      fprintf(fp," %13.6e %13.6e    ",
            cm(a,i,k)->real, cm(a,i,k)->imag);
    }
  }
  fprintf(fp,"\n");
}

void cm_display(title,a)
```

```
char *title;
complex_matrix a;
{ int i, k, l;

  printf("\nComplex Matrix %s",title);

  for (i = 0; i < a->nrows; i++)
  {
    printf("\nROW %d\n",i);

    for (k = 0, l = 0; k < a->ncols; k++, l++)
    {
      if (l == 2)
      {  l = 0;
         printf("\n");
      }

      printf("(%13.6e,%13.6e)  ",
             cm(a,i,k)->real, cm(a,i,k)->imag);
    }
  }
  printf("\n");
}

void cv_display(title, a)
char *title;
complex_matrix a;
{ int k, l;

  printf("\nComplex Vector %s\n",title);

  for (k = 0, l = 0; k < (a->ncols)*(a->nrows); k++, l++)
  {
    if (l == 2)
    {  l = 0;
       printf("\n");
    }

    printf("(%13.6e,%13.6e)  ",
           cv(a,k)->real, cv(a,k)->imag);
  }
  printf("\n");
}

/* c = a + b */
void cm_add(c,a,b)
complex_matrix c, a, b;
{ int i, k;

  for (k = 0; k < a->ncols; k++)
  { for (i = 0; i < a->nrows; i++)
    { c_add(cm(c,i,k),cm(a,i,k),cm(b,i,k));
    }
  }
}

/* c = a - b */
void cm_sub(c,a,b)
```

```
complex_matrix c, a, b;
{ int i, k;

  for (k = 0; k < a->ncols; k++)
  { for (i = 0; i < a->nrows; i++)
    { c_sub(cm(c,i,k),cm(a,i,k),cm(b,i,k));
    }
  }
}

/* c = a*b, c not the same as a or b */
void cm_mul(c,a,b)
complex_matrix c, a, b;
{ int i, k, l;

  for (k = 0; k < b->ncols; k++)
  { for (i = 0; i < a->nrows; i++)
    { c_zero(cm(c,i,k));
      for (l = 0; l < b->nrows; l++)
      { c_xy(cm(c,i,k),cm(a,i,l),cm(b,l,k));
      }
    }
  }
}

/*  cv_dotp = Re{v'*u} */
float cv_dotp(u, v)
complex_matrix u, v;
{ int i, n;
  float dotp;

  n = v->nrows * v->ncols;

  dotp = 0.0;

  for (i = 0; i < n; i++)
    dotp += c_RexyH(cv(u,i),cv(v,i));

  return dotp;
}

/* ||v||^2 */
float cv_norm2(v)
complex_matrix v;
{ int i;
  float norm2;

  norm2 = 0.0;
  for (i = 0; i < (v->nrows)*(v->ncols); i++)
  { norm2 += c_mag2(cv(v,i));
  }
  return norm2;
}

/* ||v|| */
#define cv_norm(v) sqrt(cv_norm2(v))

/* z = a*x scale by a real number */
```

```
void cm_scale(z,a,x)
complex_matrix z;
float a;
complex_matrix x;
{ int i;
  for (i = 0; i < z->nrows*z->ncols; i++)
  {  c_scale(cv(z,i),a,cv(x,i));
  }
}

/* z += a*x */
void cv_ax(z,a,x)
complex_matrix z;
float a;
complex_matrix x;
{ int i;
  for (i = 0; i < z->nrows; i++)
  {  c_ax(cv(z,i),a,cv(x,i));
  }
}

/* z = conj(x) */
void cv_conj(z,x)
complex_matrix z, x;
{ int i;
  for (i = 0; i < z->nrows*z->ncols
; i++)
  {  c_conj(cv(z,i),cv(x,i));
  }
}

/* x = (x)H */
void cm_H(x)
complex_matrix x;
{ int i,k,nr,nc;
  complex tmp;
  complex_matrix xtmp;

  nr = x->nrows;
  nc = x->ncols;

  if (nr == 1 | nc == 1)       /* vector case, quickly */
  { cv_conj(x,x);
    x->nrows = nc;
    x->ncols = nr;
  }
  else
  { xtmp = cm_copy(x);         /* matrix case, make copy, etc.. */
    x->nrows = nc;
    x->ncols = nr;
    for (i = 0; i < nr; i++)
    { for (k = 0; k < nc; k++)
      { c_conj(cm(x,k,i),cm(xtmp,i,k));
      }
    }
    cm_free(xtmp);
  }
}
```

```
/* limits real and imag. parts of x to a certain range */
float cv_limiter(x,maxv)
  complex_matrix x;
  float maxv;
/*****************/
{ int i;
  float xr, xi, max_level, absxr, absxi;

  max_level = 0.0;

  for (i = 0; i < (x->nrows * x->ncols); i++)
  { xr = cv(x,i)->real;
    xr = LIMITER(xr,maxv);
    absxr = ABS(xr);
    max_level = MAXIMUM(max_level,absxr);

    xi = cv(x,i)->imag;
    xi = LIMITER(xi,maxv);
    absxi = ABS(xi);
    max_level = MAXIMUM(max_level,absxi);

    c_assign_real(cv(x,i),xr,xi);
  }
  return max_level;
}

/* normalizes non-zero columns of x to unit magnitude */
void cm_normal(x)
complex_matrix x;
{ int i, k;
  float col_norm;

  for (k = 0; k < x->ncols; k++)
  { col_norm = 0.0;
    for (i = 0; i < x->nrows; i++)
    { col_norm += c_mag2(cm(x,i,k));
    }
    col_norm = sqrt(col_norm);

    if (col_norm > 0.0)
    { for (i = 0; i < x->nrows; i++)
      { cm(x,i,k)->real /= col_norm;
        cm(x,i,k)->imag /= col_norm;
      }
    }
  }
}

/* z += x*yH outer product */
void cv_xyH(z,x,y)
complex_matrix z,x,y;
{ int i, k;

  for (i = 0; i < z->nrows; i++)
  { for (k = 0; k < z->ncols; k++)
    { c_xyH(cm(z,i,k),cv(x,i),cv(y,k));
    }
```

```
  }
}

/* z += x*yT  outer product */
void cv_xyT(z,x,y)
complex_matrix z,x,y;
{ int i, k;

  for (i = 0; i < z->nrows; i++)
  { for (k = 0; k < z->ncols; k++)
    { c_xy(cm(z,i,k),cv(x,i),cv(y,k));
    }
  }
}

/* z -= x*yH  outer product */
void cv_mxyH(z,x,y)
complex_matrix z,x,y;
{ int i, k;

  for (i = 0; i < z->nrows; i++)
  { for (k = 0; k < z->ncols; k++)
    { c_mxyH(cm(z,i,k),cv(x,i),cv(y,k));
    }
  }
}

/* z -= x*yT  outer product */
void cv_mxyT(z,x,y)
complex_matrix z,x,y;
{ int i, k;

  for (i = 0; i < z->nrows; i++)
  { for (k = 0; k < z->ncols; k++)
    { c_mxy(cm(z,i,k),cv(x,i),cv(y,k));
    }
  }
}

/* z += a*x*yT  outer product */
void cv_axyT(z,a,x,y)
complex_matrix z;
float a;
complex_matrix x,y;
{ int i, k;

  for (i = 0; i < z->nrows; i++)
  { for (k = 0; k < z->ncols; k++)
    { c_axy(cm(z,i,k),a,cv(x,i),cv(y,k));
    }
  }
}

/* z += a*x*yH  outer product */
void cv_axyH(z,a,x,y)
complex_matrix z,x,y;
float a;
{ int i, k;
```

```
  for (i = 0; i < z->nrows; i++)
  { for (k = 0; k < z->ncols; k++)
    { c_axyH(cm(z,i,k),a,cv(x,i),cv(y,k));
    }
  }
}

/* z += xT*y  inner product, scalar result */
void cv_xTy(z,x,y)
complex z;
complex_matrix x,y;
{ int i;

  for (i = 0; i < x->nrows; i++)
  { c_xy(z,cv(y,i),cv(x,i));
  }
}

/* z += xH*y  inner product, scalar result */
void cv_xHy(z,x,y)
complex z;
complex_matrix x,y;
{ int i;

  for (i = 0; i < x->nrows; i++)
  { c_xyH(z,cv(y,i),cv(x,i));
  }
}

/* z -= xT*y  inner product, scalar result */
void cv_mxTy(z,x,y)
complex z;
complex_matrix x, y;
{ int i;

  for (i = 0; i < x->nrows; i++)
  { c_mxy(z,cv(y,i),cv(x,i));
  }
}

/* z -= xH*y  inner product, scalar result */
void cv_mxHy(z,x,y)
complex z;
complex_matrix x, y;
{ int i;

  for (i = 0; i < x->nrows; i++)
  { c_mxyH(z,cv(y,i),cv(x,i));
  }
}

/* z += a*xT*y  inner product, scalar result */
void cv_axTy(z,a,x,y)
complex z;
float a;
complex_matrix x,y;
{ int i;
```

```
  for (i = 0; i < x->nrows; i++)
  { c_axy(z,a,cv(y,i),cv(x,i));
  }
}

/* z += a*xH*y  inner product, scalar result */
void cv_axHy(z,a,x,y)
complex z;
float a;
complex_matrix x,y;
{ int i;

  for (i = 0; i < x->nrows; i++)
  { c_axyH(z,a,cv(y,i),cv(x,i));
  }
}

/*************************************************/
/* Utilities for Householder transformations.    */
/*************************************************/

/* Finds a vector v having the same number of    */
/* rows as a->ncols such that                    */
/*    a(i,:) - 2*a(i,:)*v*vH is zero after       */
/* column p. v is unit magnitude unless a is     */
/* already in this form, when it is zero.        */

complex_matrix house_v_row(a,i,p)
complex_matrix a;
int i, p;
{ int m, n;
  complex_matrix v, x;
  complex phasor0;
  float xnorm, magx0;

  n = a->ncols;
  m = n-p;

  v = cm_new(n,1,0.0,0.0);
  if (m <= 0)
  { return v;
  }

  /* Pull out x = a(i,p:(n-1))' */

  x = cm_copy_part(a,i,i,p,(n-1));
  cm_H(x);

  /* Get norm of x and |x(0)|, phasor0 = x(0)/|x(0)|. */

  xnorm = cv_norm(x);
  magx0 = c_mag(cv(x,0));

  phasor0 = c_new(1,1.0,0.0);
  if (magx0 > 0.0)
  { phasor0->real = cv(x,0)->real / magx0;
    phasor0->imag = cv(x,0)->imag / magx0;
```

```
  }

  /* Construct v in x, then assign. */

  c_ax(cv(x,0),-xnorm,phasor0);    /* x(0) = x(0) - xnorm*phasor0 */
  cm_normal(x);

  cm_assign_part(v,p,0, x,0,0, m,1);

  c_free(phasor0);
  cm_free(x);
  return v;
}

/* multiply x on left by I - 2*v*v'      */
/* xbar may not be the same vector as x  */
void house_Qx(xbar,v,x)
complex_matrix xbar, v, x;
{ complex_matrix vH, xtmp;

  if (v->nrows != x->nrows  | xbar->nrows != x->nrows
      | xbar->ncols != x->ncols)
  { return;
  }

  vH = cm_copy(v);
  cm_H(vH);

  xtmp = cm_new(1,x->ncols,0.0,0.0);
  cm_mul(xtmp,vH,x);
  cm_scale(xtmp,2.0,xtmp);

  cm_mul(xbar,v,xtmp);
  cm_sub(xbar,x,xbar);

  cm_free(vH);
  cm_free(xtmp);
}

/* multiply x on right by I - 2*v*v'     */
/* xbar may not be the same vector as x  */
void house_xQ(xbar,x,v)
complex_matrix xbar, v, x;
{ complex_matrix vH, xtmp;

  if (v->nrows != x->ncols  | xbar->nrows != x->nrows
      | xbar->ncols != x->ncols)
  { return;
  }

  vH = cm_copy(v);
  cm_H(vH);

  xtmp = cm_new(x->nrows,1,0.0,0.0);
  cm_mul(xtmp,x,v);
  cm_scale(xtmp,2.0,xtmp);

  cm_mul(xbar,xtmp,vH);
```

```
  cm_sub(xbar,x,xbar);

  cm_free(vH);
  cm_free(xtmp);
}


/*****************************************************/
/* Utilities for an LDL' complex representation  */
/*****************************************************/

typedef struct
{ int n;
  complex Lptr;
  float *Dptr;
} *LDL_decomp;

#define LDL_L(R,i,k) \
( ((R)->Lptr) + (i) + (k)*((R)->n) )

#define LDL_D(R,i) \
((R)->Dptr)[i]

/* Allocate an LDL decomposition */

LDL_decomp LDL_new(n)
int n;
{ LDL_decomp R;
  int i,k;

  R = (LDL_decomp) malloc(sizeof(*R));
  R->n = n;
  R->Lptr = c_new(n*n,0.0,0.0);
  R->Dptr = (float (*)) malloc(n * sizeof(float));

  for (i = 0; i < n; i++)
  { LDL_D(R,i) = 0.0;
    c_assign_real(LDL_L(R,i,i),1.0,0.0);
  }

  return R;
}

void LDL_free(R)
LDL_decomp R;
{ c_free(R->Lptr);
  free(R->Dptr);
  free(R);
}

void LDL_assign(Rto,Rfrom)
LDL_decomp Rto, Rfrom;
{ int i, k, n;

  n = Rfrom->n;

  for (i = 0; i < n; i++)
  { LDL_D(Rto,i) = LDL_D(Rfrom,i);
```

```
    for (k = 0; k < i; k++)
    { c_assign(LDL_L(Rto,i,k),LDL_L(Rfrom,i,k));
    }
  }
}

LDL_decomp LDL_get(fp)
FILE *fp;
{ int n, i, k, readok;
  LDL_decomp R;

  fscanf(fp,"%d",&n);

  if (n < 1)
  { printf("\nLDL_get:  n < 1!");
    return ((LDL_decomp) NULL);
  }

  R = LDL_new(n);

  for (k = 0; k < n; k++)
  { readok = fscanf(fp,"%e",&LDL_D(R,k));

    if (readok == EOF)
    { printf("LDL_get:  error reading element %d of D!",k);
      return ((LDL_decomp) NULL);
    }
  }

  for (i = 0; i < n; i++)
  { for (k = 0; k < n; k++)
    { readok = fscanf(fp,"%e%e",
                &(LDL_L(R,i,k)->real),&(LDL_L(R,i,k)->imag));

      if (readok == EOF)
      { printf("LDL_get:  error reading element (%d,%d) of L!",i,k);
        return ((LDL_decomp) NULL);
      }
    }
  }

  return R;
}

void LDL_display(title,R)
char *title;
LDL_decomp R;
{ int i, k, l;

  printf("\nLDL Decomposition %s",title);

  printf("\nDiagonal D:\n");
  for (k = 0, l = 0; k < R->n; k++, l++)
  {
    if (l == 5)
    {  l = 0;
       printf("\n");
    }
```

```
    printf("%13.6e ",LDL_D(R,k));
  }

  printf("\nStrictly Lower Triangular Part of Complex Matrix L:");
  for (i = 1; i < R->n; i++)
  {
    printf("\nROW %d\n",i);

    for (k = 0, l = 0; k < i; k++, l++)
    {
      if (l == 2)
      {  l = 0;
         printf("\n");
      }

      printf("(%13.6e,%13.6e)  ",
             LDL_L(R,i,k)->real, LDL_L(R,i,k)->imag);
    }
  }
  printf("\n");
}

/*****************************************/
/* LDL update:  Rnew = a*R + b*(x*x') */
/*****************************************/

void LDL_update(R,a,Ro,b,x)
LDL_decomp R, Ro;
float a, b;
complex_matrix x;
{ int n, i, k;
  float nu, *rho;
  complex_matrix y;
  complex S, ctmp;

  y = cm_copy(x);
  n = Ro->n;

  /* Find y such that x = Lo * y;  */

  for (i = 0; i < n; i++)
  {
    for (k = 0; k < i; k++)
    {
      /* y(i) = y(i) - Lo(i,k)*y(k);  */
      c_mxy(cv(y,i),LDL_L(Ro,i,k),cv(y,k));
    }
  }

  /* Now find rho and D. */

  rho = f_new(n,0.0);
  nu = 1.0;

  for (i = 0; i < n; i++)
  {
    LDL_D(R,i) = a * LDL_D(Ro,i)  +  b * nu * c_mag2(cv(y,i));
    LDL_D(R,i) = MAXIMUM(0.0,LDL_D(R,i));
```

```
    if (LDL_D(R,i) == 0.0)
    { rho[i] = 0.0;
    }
    else
    { rho[i] = b * nu / LDL_D(R,i);
      nu = a * nu * LDL_D(Ro,i) / LDL_D(R,i);
    }
  }

  /* Now compute L */

  S = c_new(1,0.0,0.0);
  ctmp = c_new(1,0.0,0.0);

  for (i = 1; i < n; i++)
  {
    /*  L(i,i-1) = Lo(i,i-1) + y(i)*y(i-1)'*rho[i-1];  */
    c_assign(LDL_L(R,i,i-1),LDL_L(Ro,i,i-1));
    c_axyH(LDL_L(R,i,i-1),rho[i-1],cv(y,i),cv(y,i-1));

    c_assign_real(S,0.0,0.0);

    for (k = i-2; k >= 0; k--)
    {
      /* S = S + Lo(i,k+1)*y(k+1);  */
      c_xy(S,LDL_L(Ro,i,k+1),cv(y,k+1));

      /* L(i,k) = Lo(i,k) + (S + y(i))*y(k)'*rho(k);  */
      c_assign(LDL_L(R,i,k),LDL_L(Ro,i,k));
      c_add(ctmp,S,cv(y,i));
      c_axyH(LDL_L(R,i,k),rho[k],ctmp,cv(y,k));
    }
  }

  /* Free allocated resources. */

  c_free(S);
  c_free(ctmp);
  f_free(rho);
  cm_free(y);

} /* end LDL_update */

/*************************************/
/* LDL inversion:  u = inv(LDL')*v; */
/*************************************/

void LDL_inv(u, R, v)
complex_matrix u, v;
LDL_decomp R;
{ int i, k, n;
  float maxu2;

  n = R->n;

  /* Back substitute to get u = inv(L)*v */
```

```
  for (i = 0; i < n; i++)
  {
    c_assign(cv(u,i),cv(v,i));

    for (k = 0; k < i; k++)
    {
      c_mxy(cv(u,i),cv(u,k),LDL_L(R,i,k));
    }
  }

  /* Use pseudo-inverse of d */

  maxu2 = c_mag2(cv(u,0));

  for (i = 1; i < n; i++)
  { maxu2 = MAXIMUM(maxu2,c_mag2(cv(u,i)));
  }

  for (i = 0; i < n; i++)
  { if (c_mag2(cv(u,i)) > C_TOLER*maxu2)
    { c_scale(cv(u,i),(1.0/LDL_D(R,i)),cv(u,i));
    }
  }

  /* Back substitute to get u */

  for (i = (n-1); i >= 0; i--)
  {
    for (k = (n-1); k > i; k--)
    {
      c_mxyH(cv(u,i),cv(u,k),LDL_L(R,k,i));
    }
  }
}


/*************************************************/
/* LDL Pseudo Inverse constructor.               */
/* R = dP2*eye + P'*P                            */
/*************************************************/

void LDL_pseudo(R,P,dP2)
LDL_decomp R;
complex_matrix P;
float dP2;
{ int i, k, n;
  LDL_decomp Rtmp;
  complex_matrix ptmp;

  n = R->n;
  Rtmp = LDL_new(n);
  ptmp = cm_new(n,1,0.0,0.0);

  for (i = 0; i < n; i++)
  { LDL_D(Rtmp,i) = dP2;
  }

  /* Now accumulate the rows of P */
```

```
  for (i = 0; i < n; i++)
  {
      for (k = 0; k < n; k++)
      { c_conj(cv(ptmp,k),cm(P,i,k));
      }

    LDL_update(R,1.0,Rtmp,1.0,ptmp);
    LDL_assign(Rtmp,R);
  }

  cm_free(ptmp);
  LDL_free(Rtmp);
}

/* ainv = aH * inv(a * aH) */
void cm_inv2(ainv,a)
complex_matrix ainv, a;
{ complex_matrix u, v, aHinv;
  LDL_decomp aaH, aaHtmp;
  int nr, nc, i;

  nr = a->nrows;
  nc = a->ncols;
  aHinv = cm_copy(a);

  aaH = LDL_new(nr);
  aaHtmp = LDL_new(nr);

  u = cm_new(nr,1,0.0,0.0);
  v = cm_new(nr,1,0.0,0.0);

  /*  Create aaH = a * aH as an LDL decomposition. */

  for (i = 0; i < nc; i++)
  { cm_assign_col(u,0,a,i);
    LDL_assign(aaHtmp,aaH);
    LDL_update(aaH,1.0,aaHtmp,1.0,u);
  }

  /* Back substitute aHinv(:,i) = inv(a*aH) * a(:,i) */

  for (i = 0; i < nc; i++)
  { cm_assign_col(u,0,a,i);
    LDL_inv(v,aaH,u);
    cm_assign_col(aHinv,i,v,0);
  }

  cm_H(aHinv);
  cm_assign(ainv,aHinv);

  LDL_free(aaH);
  LDL_free(aaHtmp);
  cm_free(u);
  cm_free(v);
  cm_free(aHinv);
}
```

```
void LDL_eye(R,diag_val)
LDL_decomp R;
float diag_val;
{ int i,k;

  for (i = 0; i < R->n; i++)
  { LDL_D(R,i) = diag_val;

    for (k = 0; k < i; k++)
    { c_zero(LDL_L(R,i,k));
    }
  }
}
```

Floating Point Utilities (floatutl.h)

```
/* floating point vector utilities */

float *f_new(n,fill_value)
int n;
float fill_value;
{ int i;
  float *a;
  a = (float *) malloc(n*sizeof(float));
  for (i = 0; i < n; i++)
    a[i] = fill_value;
  return a;
}

void f_free(p)
float *p;
{ free(p);
}

void f_display(title,p,n)
char *title;
float *p;
int n;
{ int i,l;

  printf("\nfloat vector %s\n",title);

  for (l = 0, i = 0; i < n; i++)
  {
     printf("%13.6e  ",p[i]);

     l++;
     if (l == 5)  l = 0;
  }
}
```

A/D Interface (ad_lib.h, ad_lib.c)

ad_lib.h:

```
/*---------------------------------------*/
/* Function prototypes for ad_lib_gen    */
/*---------------------------------------*/

void ad_setup(start_channel,nchannels);
void ad_convert(float_buf_ptr,nchannels);
```

ad_lib.c:

```
/*-------------------------------------------*/
/* File to define routines for ad control.   */
/* L. Davis  5/9/96                          */
/*-------------------------------------------*/

/* type definitions for various word sizes */

typedef unsigned char onebyte;
typedef unsigned short int twobyte;
typedef unsigned int fourbyte;

/* base addresses, etc. for 601 */

#define AD_BASE 0xF0FA0000
#define AD_DPR_LEN 0xFFFF
#define AD_CMD_OFFSET 0xFFF0
#define AD_WAKE_OFFSET 0xFFEE
#define AD_ZERO_CHAN 192

twobyte *hststa = (twobyte *) (AD_BASE + 0xFFF6);
twobyte *adstat = (twobyte *) (AD_BASE + 0xFFF4);
twobyte *execcmd = (twobyte *) (AD_BASE + 0xFFFC);

/* routine addresses for the 601 */

#define DFSBUF ((fourbyte) 0x12E)
#define DFDBUF ((fourbyte) 0x134)
#define SLSBUF ((fourbyte) 0x13A)
#define SLWTEM ((fourbyte) 0x14C)
#define SLDPXF ((fourbyte) 0x15E)
#define XLCABF ((fourbyte) 0x170)
#define FASNSC ((fourbyte) 0x17C)
#define STSNSC ((fourbyte) 0x188)
#define STSNSN ((fourbyte) 0x1BE)

/* define low level tasks */

/*-------------------------------------*/
/* Allow permission to read/write      */
/* code is in assembler at bottom.     */
/* sys_permit(start_addr,#bytes);      */
/*-------------------------------------*/
static void sys_permit();
```

```
#asm:
sys_permit
    move.l a2,-(sp)     *save address
    movea.l d0,a2       *base address to a2 (argument 1)
    move.l d1,d0        *offset to highest value (arg 2)
    moveq.l #7,d1
    os9     F$Permit    *allow writing to these addresses/reading
    movea.l (sp)+,a2    *restore a2
    rts
#endasm

/*-----------------------------------*/
/* Clear the DPR area out.           */
/*-----------------------------------*/
ad_clear_dpr()
{ onebyte *bptr;

  for (bptr = (onebyte *) AD_BASE;
       bptr < (onebyte *) (AD_BASE + AD_DPR_LEN); bptr++)
  { *bptr = (onebyte) 0;
  }
}

/*-----------------------------------*/
/* wait for acknowledgement of cmd   */
/*-----------------------------------*/

void ad_wait_ack()
{ int i = 0;

  while (!(0x0001 & *hststa))        /* check bit zero */
  { i++; i++; i--; i--;        /* burn a few ops */
  }
}

void ad_reset_ack()
{ *hststa = (*hststa & 0xFFFE);
}

/*-----------------------------------*/
/* wait for afb0 or afb0 error       */
/*-----------------------------------*/

void ad_wait_afb0()
{ int i = 0;

  while ((0x0050 & *hststa) == 0)        /* check bit zero */
  { i++; i++; i--; i--;        /* burn a few ops */
  }

  if ((0x0010 & *hststa) != 0)
  { printf("ad_wait_afb0:  error in afb0 execution!");
  }
}

/*-----------------------------------*/
/* wait for afb1 or afb1 error       */
/*-----------------------------------*/
```

```
void ad_wait_afb1()
{ int i = 0;

  while ((0x00A0 & *hststa) == 0)        /* check bit zero */
  { i++; i++; i--; i--;        /* burn a few ops */
  }

  if ((0x0020 & *hststa) != 0)
  { printf("ad_wait_afb1:  error in afb1 execution!");
  }
}

/*-------------------------------------*/
/* execute command                     */
/*-------------------------------------*/
void ad_execute()
{ *hststa = (*hststa & 0xFFFE);      /* reset ack */
  *execcmd = (twobyte) 0;            /* any write causes execution */
}

/*-------------------------------------*/
/* transfer string to DPR cmd          */
/* return new address                  */
/*-------------------------------------*/

onebyte *ad_cmds_move(bptr,str)
onebyte *bptr;
char *str;
{ int i;

  for (i = strlen(str)-1; i >= 0; i--)
  { *(--bptr) = (onebyte) str[i];
  }
  return bptr;
}

/*-------------------------------------*/
/* transfer fourbyte data to DPR cmd.  */
/* return new buffer address.          */
/* alignment may not be even.          */
/*-------------------------------------*/

onebyte *ad_cmd4_move(bptr,data)
onebyte *bptr;
fourbyte data;
{
  *(--bptr) = (onebyte) (data & 0x000000FF);
  data = data >> 8;
  *(--bptr) = (onebyte) (data & 0x000000FF);
  data = data >> 8;
  *(--bptr) = (onebyte) (data & 0x000000FF);
  data = data >> 8;
  *(--bptr) = (onebyte) (data & 0x000000FF);

  return bptr;
}
```

```
/*----------------------------------------*/
/* View memory byte by byte               */
/*----------------------------------------*/

void view_mem(base,offset,nbytes)
onebyte *base;
int offset, nbytes;
{ onebyte *ptr;
  static char *ctrl_str[32];
  char c;

  ctrl_str[0] = "<nul>";
  ctrl_str[1] = "<soh>";
  ctrl_str[2] = "<stx>";
  ctrl_str[3] = "<etx>";
  ctrl_str[4] = "<eot>";
  ctrl_str[5] = "<enq>";
  ctrl_str[6] = "<ack>";
  ctrl_str[7] = "<bel>";
  ctrl_str[8] = "<bs> ";
  ctrl_str[9] = "<ht> ";
  ctrl_str[10] = "<lf> ";
  ctrl_str[11] = "<vt> ";
  ctrl_str[12] = "<ff> ";
  ctrl_str[13] = "<cr> ";
  ctrl_str[14] = "<so> ";
  ctrl_str[15] = "<si> ";
  ctrl_str[16] = "<dle>";
  ctrl_str[17] = "<dc1>";
  ctrl_str[18] = "<dc2>";
  ctrl_str[19] = "<dc3>";
  ctrl_str[20] = "<dc4>";
  ctrl_str[21] = "<nak>";
  ctrl_str[22] = "<syn>";
  ctrl_str[23] = "<etb>";
  ctrl_str[24] = "<can>";
  ctrl_str[25] = "<em> ";
  ctrl_str[26] = "<sub>";
  ctrl_str[27] = "<esc>";
  ctrl_str[28] = "<fs> ";
  ctrl_str[29] = "<gs> ";
  ctrl_str[30] = "<rs> ";
  ctrl_str[31] = "<us> ";

  printf("\n");
  printf("\n+----------------------------+");
  printf("\n|       MEMORY IMAGE         |");
  printf("\n+------------+------+-------+");
  printf("\n|  address   | hex  | char  |");
  printf("\n+------------+------+-------+");

  for (ptr = base+offset; ptr < (base+offset+nbytes); ptr++)
  { printf("\n|  %8.8x  |  %2.2x  ",(fourbyte) ptr,*ptr);
    c = *ptr;
    if (c < 32)
    { printf("| %5s |",ctrl_str[(int) c]);
    }
    else if (c >= 32 & c <= 126)
```

```
    { printf("|   %c    |",c);
    }
    else if (c == 127)
    { printf("|  del   |");
    }
    else
    { printf("|no char|");
    }
  }

  printf("\n+------------+------+-------+");
  printf("\n");
}

/*-------------------------------------------------------*/
/* display hststa status                                 */
/*-------------------------------------------------------*/
void view_hststa()
{ int bits[16], i;
  twobyte status_word, one = 0x1;

  status_word = *hststa;

  for (i = 0; i < 16; i++)
  { bits[i] = ((one << i) & status_word) != 0;
  }

  printf("\n");
  printf("\n+------------------ hststa status high byte --------------
------+");
  printf("\n| func1 | func0 | func1 | func0 | cntrl |  not   |  not   |
downl |");
  printf("\n| act.  | act.  | def.  | def.  |   S   | used |  used |
done  |");
  printf("\n|   %x    |   %x    |   %x    |   %x    |",
                bits[15],bits[14],bits[13],bits[12]);
  printf("   %x    |   %x    |   %x    |   %x    |",
                bits[11],bits[10],bits[9],bits[8]);
  printf("\n+------------------ hststa status low byte ---------------
------+");
  printf("\n| func1 | func0 | func1 | func0 |  bad   |  cmd   | downl |
host  |");
  printf("\n| compl | compl | error | error |  cmd   | error | error |
ack.  |");
  printf("\n|   %x    |   %x    |   %x    |   %x    |",
                bits[7],bits[6],bits[5],bits[4]);
  printf("   %x    |   %x    |   %x    |   %x    |",
                bits[3],bits[2],bits[1],bits[0]);
  printf("\n+-------------------------------------------------------
------|");
}

/*-------------------------------------------------------*/
/* display adstat semantics                              */
/*-------------------------------------------------------*/
void view_adstat()
{ twobyte status_word;
  int bits[16], i;
```

```
  twobyte one = 0x1;

  status_word = *adstat;

  for (i = 0; i < 16; i++)
  { bits[i] = ((one << i) & status_word) != 0;
  }

  printf("\n");
  printf("\n+--------------------- adstat high byte -----------------
------+");
  printf("\n| exec. |  not  | pacer |  not  |wait on| local |  not
|use dbl|");
  printf("\n| act.  | used  | def.  | used  | host  | buffer|  used |
buffer|");
  printf("\n|   %x   |   %x   |   %x   |   %x   |",
                bits[15],bits[14],bits[13],bits[12]);
  printf("   %x   |   %x   |   %x   |   %x   |",
                bits[11],bits[10],bits[9],bits[8]);
  printf("\n+---------------------- adstat low byte -----------------
------+");
  printf("\n| scan  | conv. |  not  | buf 2 | buf 1 | buf 2 | buf 1 |
which |");
  printf("\n| done  | done  | used  | def.  | def.  | rdy   | rdy   |
buf.  |");
  printf("\n|   %x   |   %x   |   %x   |   %x   |",
                bits[7],bits[6],bits[5],bits[4]);
  printf("   %x   |   %x   |   %x   |   %x   |",
                bits[3],bits[2],bits[1],bits[0]);
  printf("\n+-------------------------------------------------------
------|");
}

/*============================================================*/
/*                  HIGHER LEVEL ROUTINES                     */
/*============================================================*/

/*---------------------------------*/
/* wake up the executive           */
/*---------------------------------*/

void ad_wake_up()
{ onebyte *bptr;

  /* allow DPR access */

  sys_permit((fourbyte) AD_BASE,(fourbyte) AD_DPR_LEN);
  ad_clear_dpr();

  /* write wakeup commands into the DPR */

  bptr = (onebyte *) (AD_BASE + AD_WAKE_OFFSET);
  bptr = ad_cmds_move(bptr,"EXEC ON\r");

  /* execute the command and try again. */

  tsleep(20);
```

```
  *hststa = (twobyte) 0;
  *adstat = (twobyte) 0;

  ad_execute();
  tsleep(100);

  /* write wakeup commands into the DPR */

  bptr = (onebyte *) (AD_BASE + AD_WAKE_OFFSET);
  bptr = ad_cmds_move(bptr,"EXEC ON\r");

  /* execute the command and await acknowledgement */

  tsleep(20);

  *hststa = (twobyte) 0;
  *adstat = (twobyte) 0;

  ad_execute();
  tsleep(1);

  ad_wait_ack();

  /* check for errors */

  if (*hststa != 0x0001)
  { printf("\nad_wake_up:  hststa = %x (hex), wakeup
failed!",*hststa);
    return;
  }

  if (*adstat != 0x8000)
  { printf("\nad_wake_up:  adstat = %x (hex), wakeup
failed!",*adstat);
    return;
  }
}

/*-----------------------------------------*/
/* Execute a single command to ad          */
/*-----------------------------------------*/

void ad_do_it(cmd_str)
char *cmd_str;
{ onebyte *bptr;

  bptr = (onebyte *) (AD_BASE + AD_CMD_OFFSET);
  bptr = ad_cmds_move(bptr,cmd_str);
  bptr = ad_cmds_move(bptr,"ENDC");

  tsleep(10);
  ad_execute();
  tsleep(1);
  ad_wait_ack();
}

/*-----------------------------------------*/
/* Set up a/d for scans                    */
```

```
/*-----------------------------------------*/
void ad_setup(start_channel,nchannels)
int start_channel, nchannels;
{ onebyte *bptr;

  /* Wake up the a/d converter */

  ad_wake_up();

  /* Define "application function block (afb)" 0 */

  bptr = (onebyte *) (AD_BASE + AD_CMD_OFFSET);
  bptr = ad_cmds_move(bptr,"BEG0");

  /* Define a single buffer in the dual-port memory for data. */
  /* Set base offset zero, channel count.                     */

  bptr = ad_cmd4_move(bptr,DFSBUF);
  bptr = ad_cmd4_move(bptr,(fourbyte) 0);
  bptr = ad_cmd4_move(bptr,(fourbyte) nchannels);

  /* Set single buffer transfers. */

  bptr = ad_cmd4_move(bptr,SLSBUF);

  /* Make a/d wait until the data ready flag is reset by  */
  /* the host before doing the next conversion.           */

  bptr = ad_cmd4_move(bptr,SLWTEM);

  /* Transmit data directly to the dual port RAM instead of */
  /* to local a/d memory                                    */

  bptr = ad_cmd4_move(bptr,SLDPXF);

  /* Finish definition of AFB0, let the a/d digest it.  */

  bptr = ad_cmds_move(bptr,"ENDF");
  bptr = ad_cmd4_move(bptr,(fourbyte) 0);    /* insurance policy */

  tsleep(20);
  ad_execute();
  tsleep(1);
  ad_wait_ack();

  /* Go ahead and process these setup commands once. */

  ad_do_it("SEL0");      /* select afb0  */
  ad_do_it("STRS");      /* execute once */
  ad_wait_afb0();        /* wait until complete */

  /* Redefine AFB0 */

  bptr = (onebyte *) (AD_BASE + AD_CMD_OFFSET);
  bptr = ad_cmds_move(bptr,"BEG0");

  /* Use "fast" single scans. */
```

```
  bptr = ad_cmd4_move(bptr,FASNSC);
  bptr = ad_cmd4_move(bptr,((fourbyte) AD_ZERO_CHAN + start_channel));
  bptr = ad_cmd4_move(bptr,
         ((fourbyte) (AD_ZERO_CHAN + start_channel + nchannels - 1)));

  /* Finish afb0 definition, let a/d digest. */

  bptr = ad_cmds_move(bptr,"ENDF");
  bptr = ad_cmd4_move(bptr,(fourbyte) 0);

  tsleep(20);
  ad_execute();
  tsleep(1);
  ad_wait_ack();

  /* Now execute this continually until a software a/d reset, etc. */

  ad_do_it("SEL0");
  ad_do_it("CONT");

  /* Wait for a/d to get the message. */

  ad_wait_afb0();
}

/*-------------------------------------------------------------*/
/* Start conversion, wait and retrieve                         */
/* bufptr    -- pointer to array of floats to put voltages in. */
/* nchannels -- number of voltages to collect.                 */
/*-------------------------------------------------------------*/
void ad_convert(bufptr,nchannels)
float *bufptr;
int nchannels;
{ short int *adptr, i;

  /* Reset the data ready bit of adstat to start the scan. */
  /* Wait for it to reassert when the scan is done.        */

  *adstat &= (0xFD);

  while ((*adstat & 0x02) == 0)
  { i = 0; i++; i--;                    /* burn some time */
  }

  /* fetch the data from the buffer, converting to float voltages */

  for (i = 0, adptr = (short int *) AD_BASE; i < nchannels; i++)
  { *(bufptr++) =  *(adptr++) / 3276.8;
  }

}
```

D/A Interface (da_lib.h, da_lib.c)

da_lib.h:

```
/*=========================================*/
/* Function prototypes for PSI's da drivers */
/*=========================================*/

void da0_setup();    /* call once to set up a board */
void da1_setup();
void da2_setup();

/* voltages   -- (float *) pointer to voltages between -10.0 & 10.0
*/
/* start_chan -- (int) to which channel to write the first voltage
*/
/* nchan      -- (int) number of successive channels to write
*/

void da0_convert(voltages,start_chan,nchan);
void da1_convert(voltages,start_chan,nchan);
void da2_convert(voltages,start_chan,nchan);
```

da_lib.c:

```
/*=====================================*/
/* PSI's da driver file.               */
/*=====================================*/

/*-----------------------------------------------------------*/
/* set da addresses                                          */
/*-----------------------------------------------------------*/

#define DA0_BASE 0xF0000000
#define DA1_BASE 0xF0000100
#define DA2_BASE 0xF0000200

#define CMD_OFFSET 0
#define CHAN_OFFSET 2
#define DATA_OFFSET 4
#define UPDATE_OFFSET 6

static short int *da0_cmd = (DA0_BASE + CMD_OFFSET);
static short int *da1_cmd = (DA1_BASE + CMD_OFFSET);
static short int *da2_cmd = (DA2_BASE + CMD_OFFSET);

static short int *da0_chan = (DA0_BASE + CHAN_OFFSET);
static short int *da1_chan = (DA1_BASE + CHAN_OFFSET);
static short int *da2_chan = (DA2_BASE + CHAN_OFFSET);

static short int *da0_data = (DA0_BASE + DATA_OFFSET);
static short int *da1_data = (DA1_BASE + DATA_OFFSET);
static short int *da2_data = (DA2_BASE + DATA_OFFSET);

static short int *da0_update = (DA0_BASE + UPDATE_OFFSET);
static short int *da1_update = (DA1_BASE + UPDATE_OFFSET);
static short int *da2_update = (DA2_BASE + UPDATE_OFFSET);

/* Set mode to immediate conversion on write-through    */
```

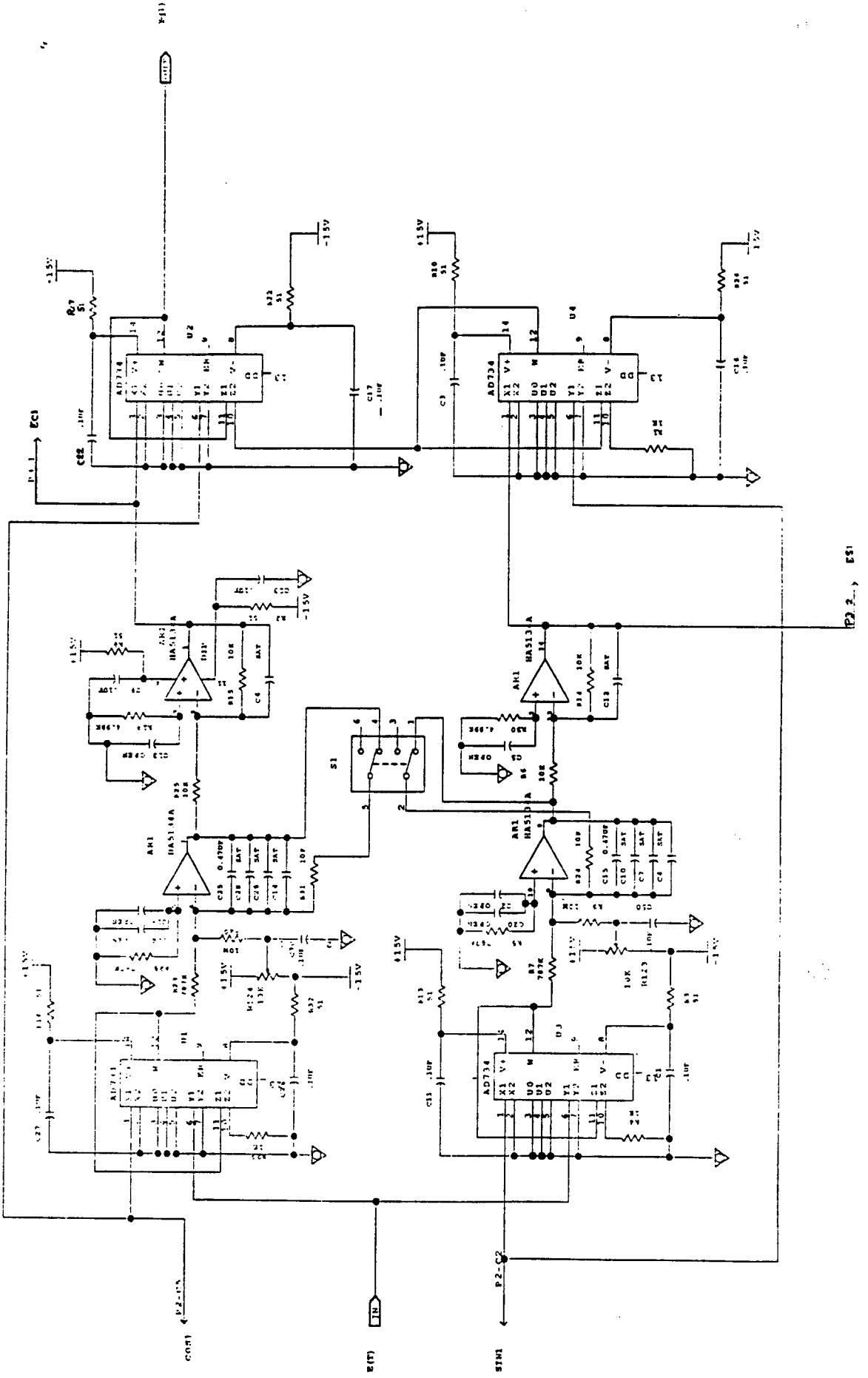
```
/* with no auto-incrementing of the address register    */
/* must write something to the update register to cause */
/* a conversion.                                        */

#define MODE 0x0000

/* Keep things in range */

#define MAXDA 2048
#define DASENS (((float) MAXDA) / 10.0)
#define damax(a,b) (((a) < (b)) ? (b) : (a))
#define damin(a,b) (((a) > (b)) ? (b) : (a))
#define da_encode(volts) (MAXDA + \
  (short int) (DASENS * damin(10.0,damax(-10.0,volts))))

/*-------------------------------------------------------*/
/* system memory access permission routine.              */
/*-------------------------------------------------------*/

static void sys_permit();

#asm:
sys_permit
   move.l       a2,-(sp)
   movea.l      d0,a2
   move.l       d1,d0
   moveq.l      #7,d1
   os9                F$Permit
   movea.l      (sp)+,a2
   rts
#endasm

/*-------------------------------------------------------*/
/* set up the d/a by writing each with the mode          */
/*-------------------------------------------------------*/
void da0_setup()
{ sys_permit((unsigned int) DA0_BASE, 0xF);
  *da0_cmd = MODE;
}

void da1_setup()
{ sys_permit((unsigned int) DA1_BASE, 0xF);
  *da1_cmd = MODE;
}

void da2_setup()
{ sys_permit((unsigned int) DA2_BASE, 0xF);
  *da2_cmd = MODE;
}

/*-------------------------------------------------------*/
/* Send out a buffer full to da0 for conversion.         */
/* bufptr   -- pointer to buffer of float voltages to    */
/*             write in [-10.0,10.0]                     */
/*-------------------------------------------------------*/
void da0_convert(bufptr, start_chan, nchan)
float *bufptr;
int start_chan, nchan;
```

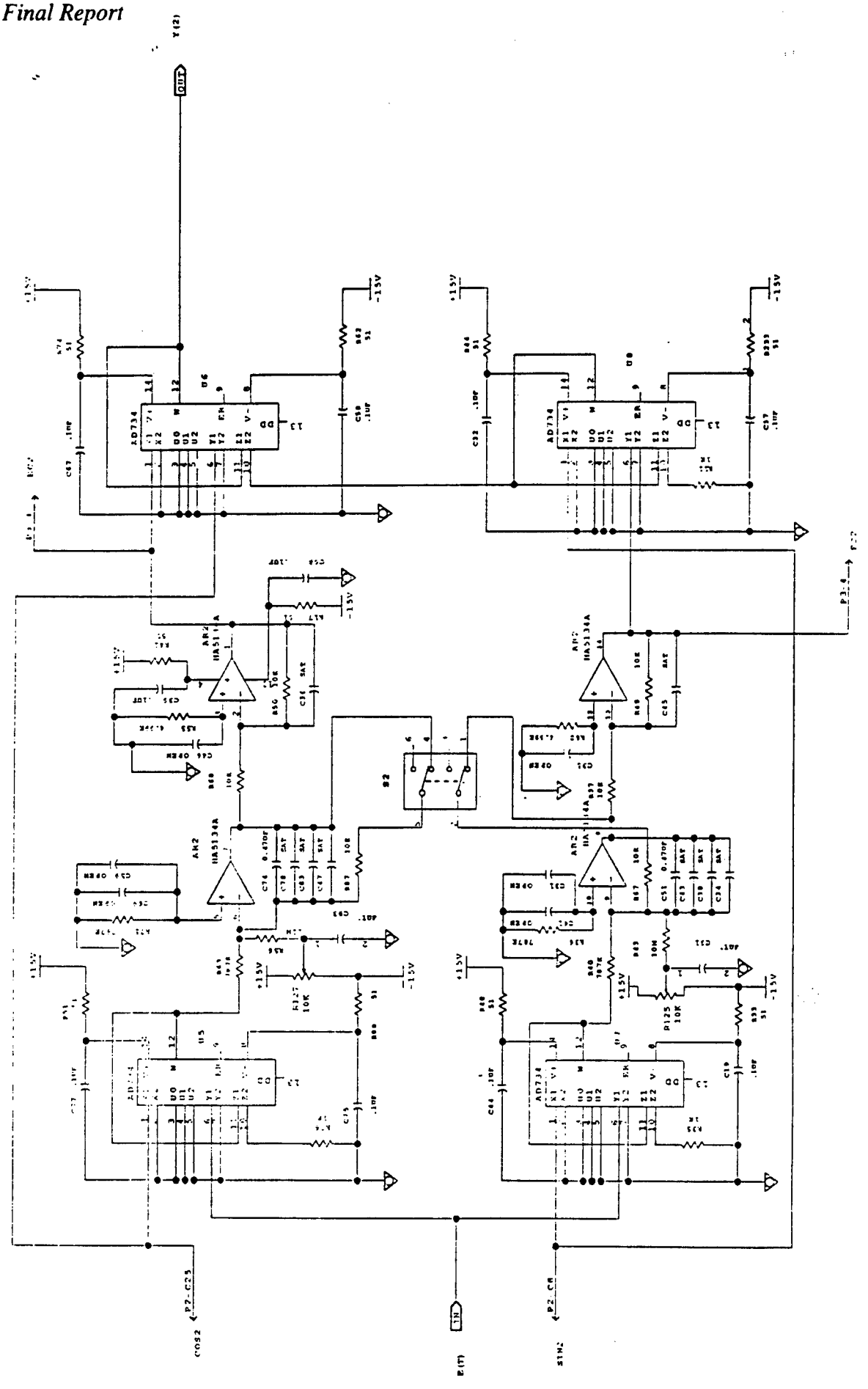
```
{ int i;
  float volts;
  short int value;

  /* Convert to an in-range integer for writing to da, then write */

  for (i = start_chan; i < start_chan + nchan; i++)
  { volts = *bufptr++;
    value = da_encode(volts);
    *da0_chan = i;
    *da0_data = value;
  }
  *da0_update = 1;              /* write anything to convert */
}

/*-------------------------------------------------------------*/
/* Send out a buffer full to da1 for conversion.               */
/* bufptr   -- pointer to buffer of float voltages to          */
/*             write in [-10.0,10.0]                           */
/*-------------------------------------------------------------*/
void da1_convert(bufptr, start_chan, nchan)
float *bufptr;
int start_chan, nchan;
{ int i;
  float volts;
  short int value;

  /* Convert to an in-range integer for writing to da, then write */

  for (i = start_chan; i < start_chan + nchan; i++)
  { volts = *bufptr++;
    value = da_encode(volts);
    *da1_chan = i;
    *da1_data = value;
  }
  *da1_update = 1;              /* write anything to convert */
}

/*-------------------------------------------------------------*/
/* Send out a buffer full to da2 for conversion.               */
/* bufptr   -- pointer to buffer of float voltages to          */
/*             write in [-10.0,10.0]                           */
/*-------------------------------------------------------------*/
void da2_convert(bufptr, start_chan, nchan)
float *bufptr;
int start_chan, nchan;
{ int i;
  float volts;
  short int value;

  /* Convert to an in-range integer for writing to da, then write */

  for (i = start_chan; i < start_chan + nchan; i++)
  { volts = *bufptr++;
    value = da_encode(volts);
    *da2_chan = i;
    *da2_data = value;
  }
```

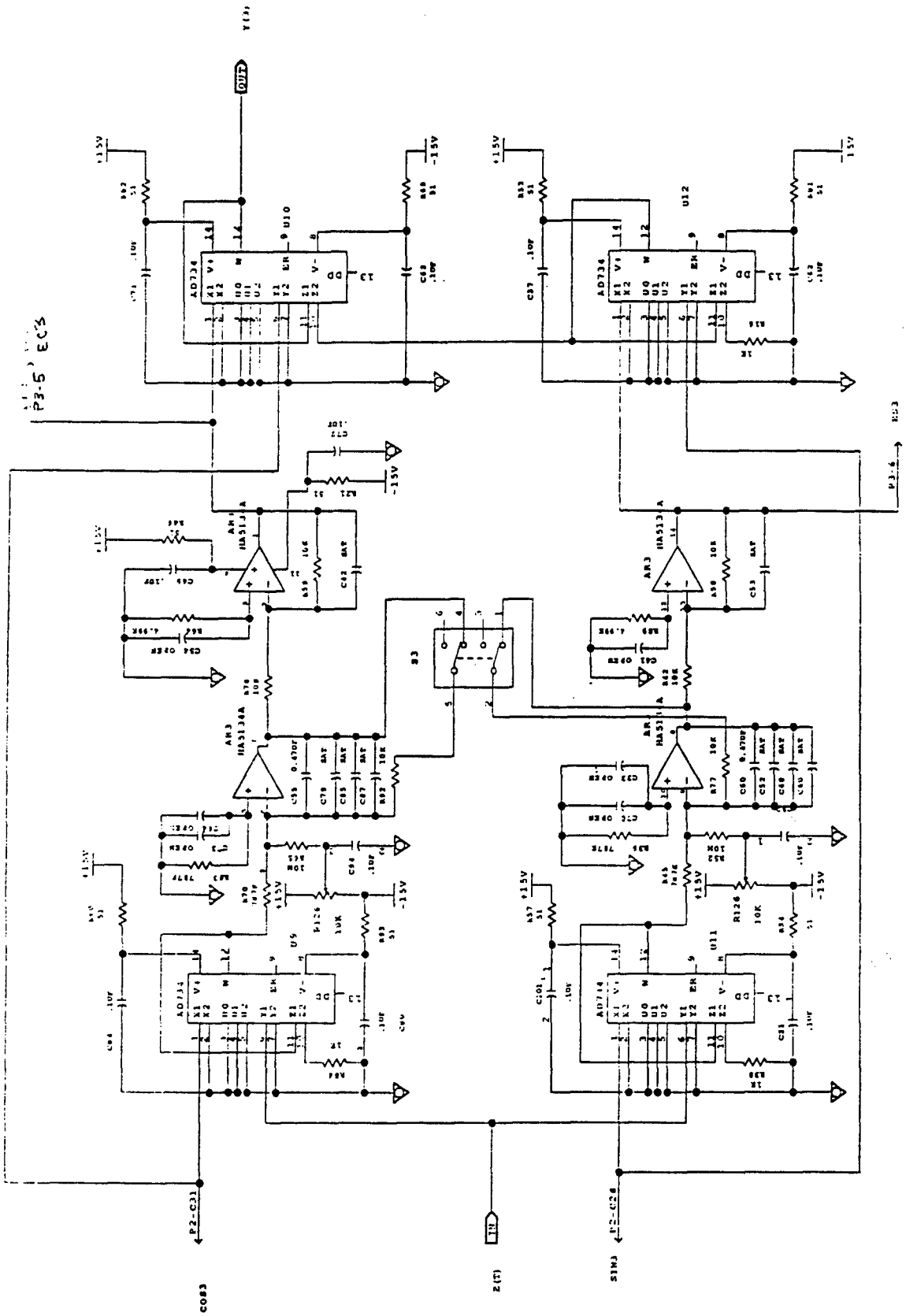
```
  *da2_update = 1;                /* write anything to convert */
}
```

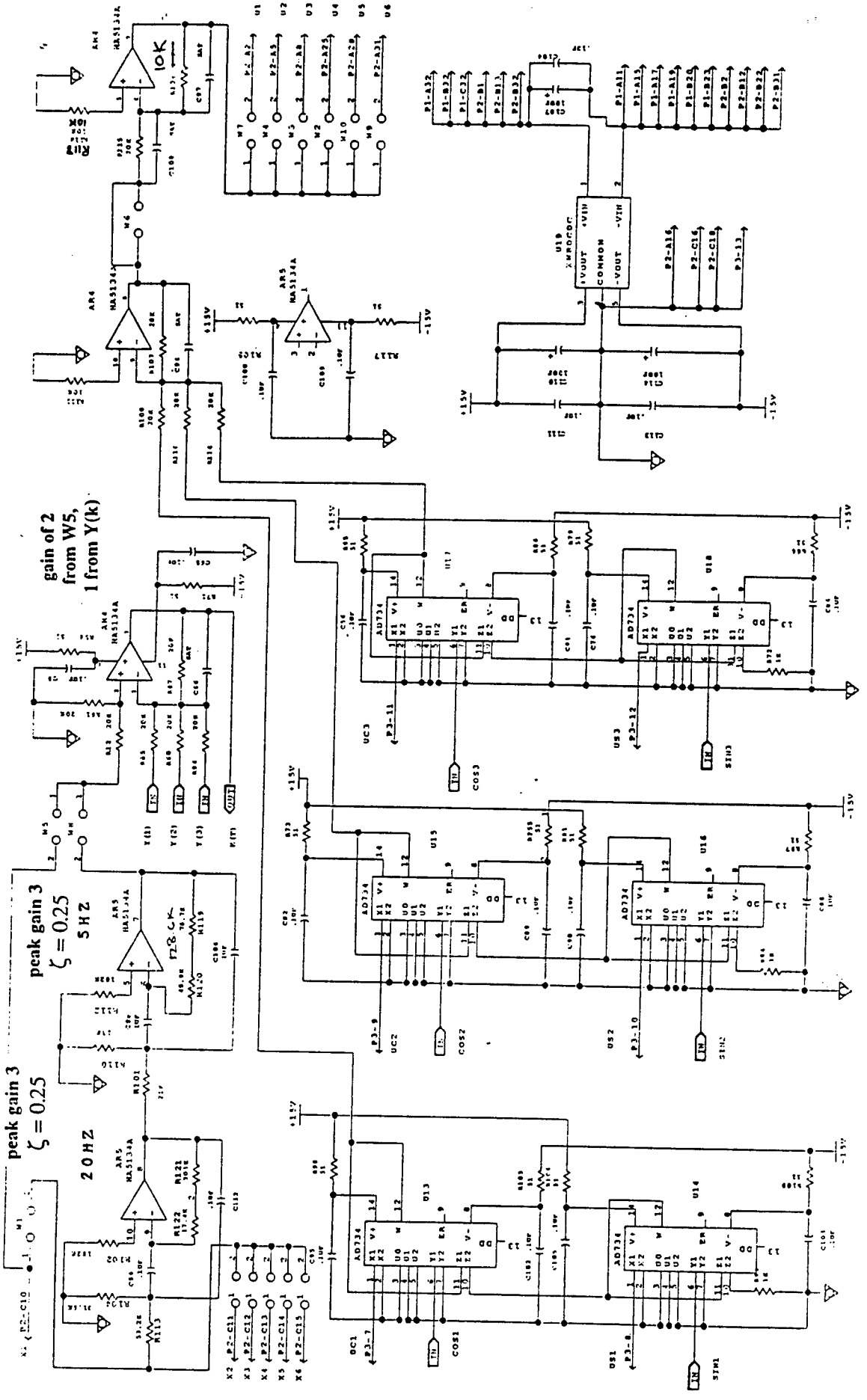
Appendix B. Electrical Schematics

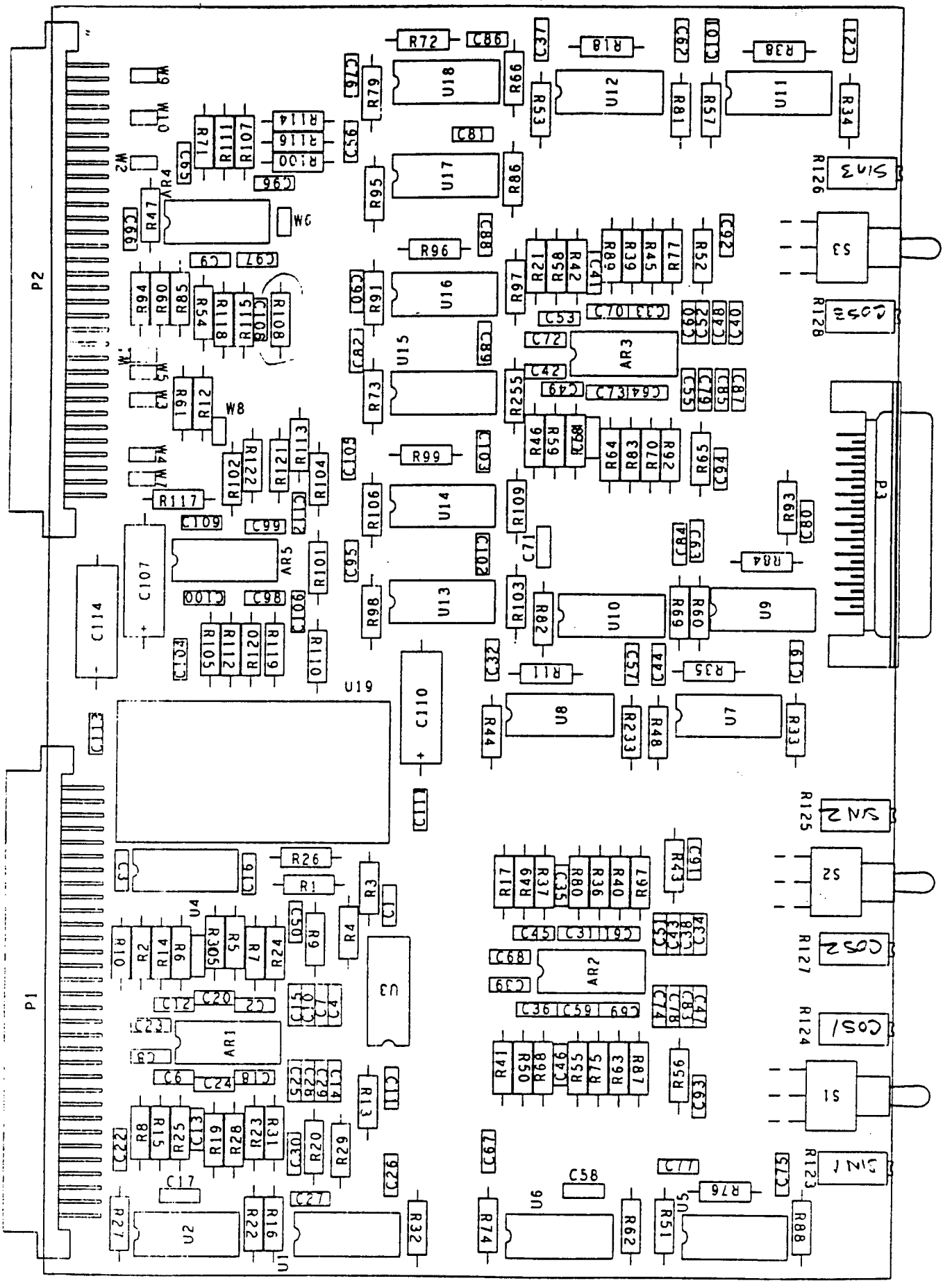
Analog Demodulator Cards



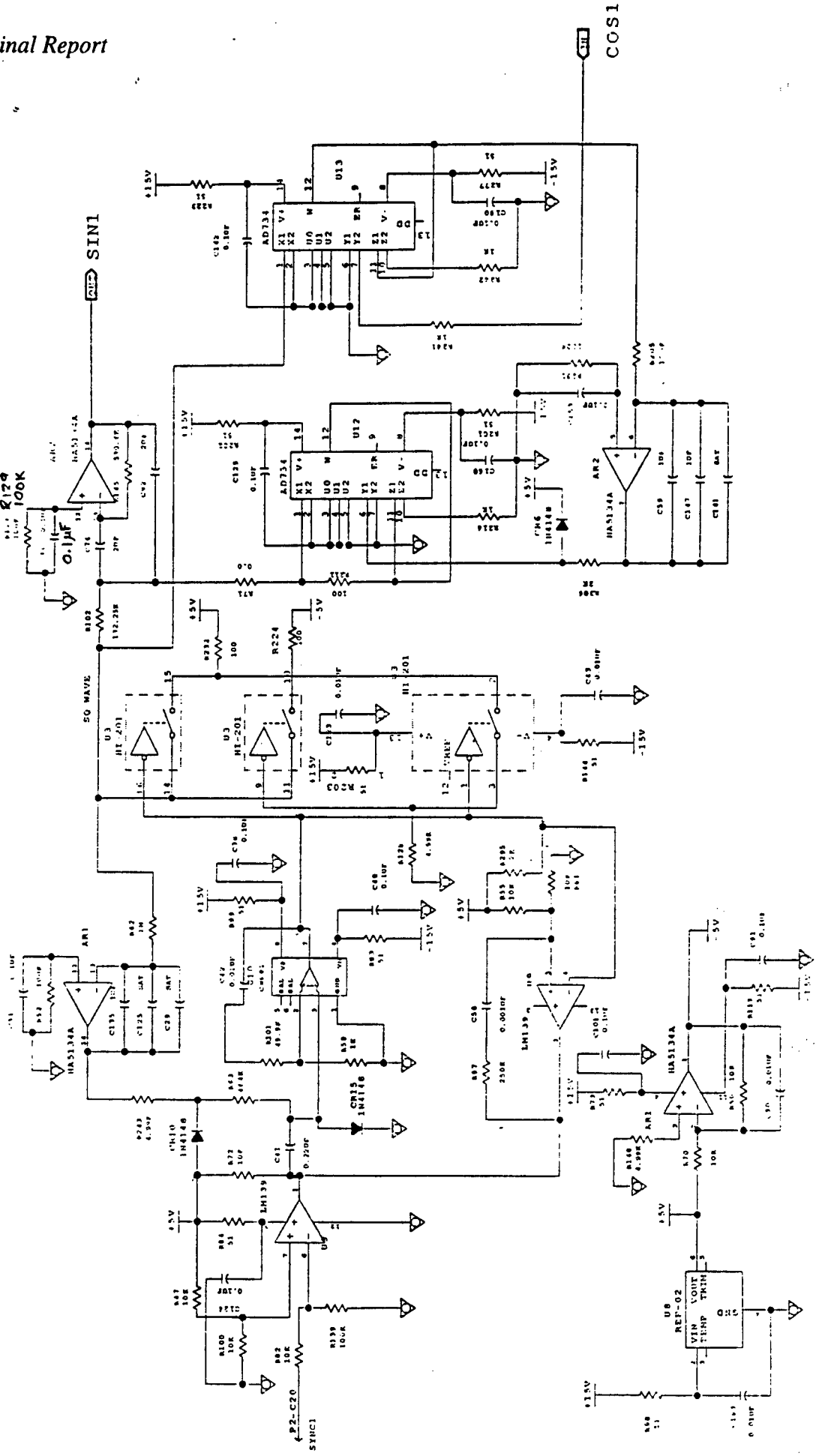


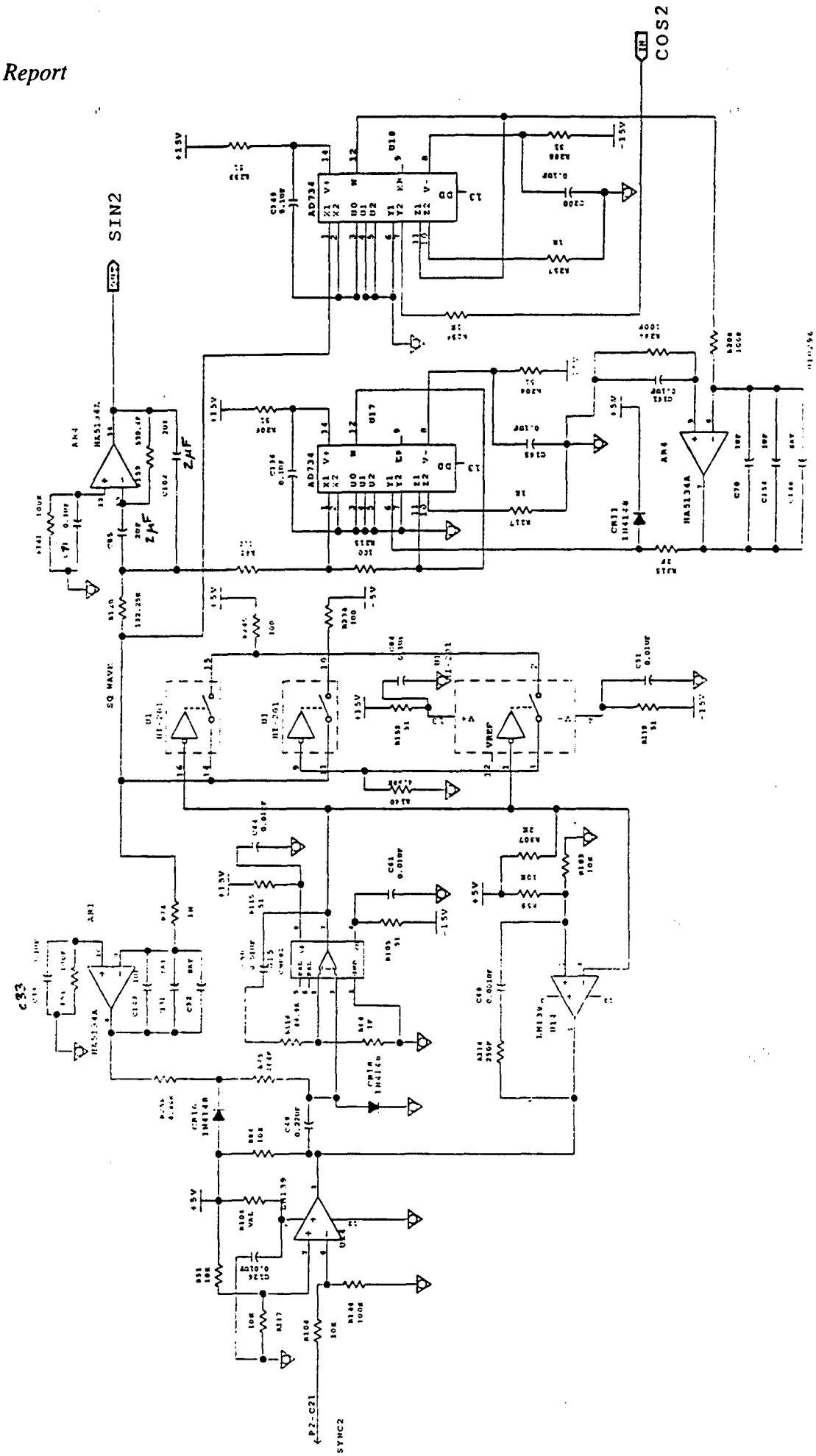


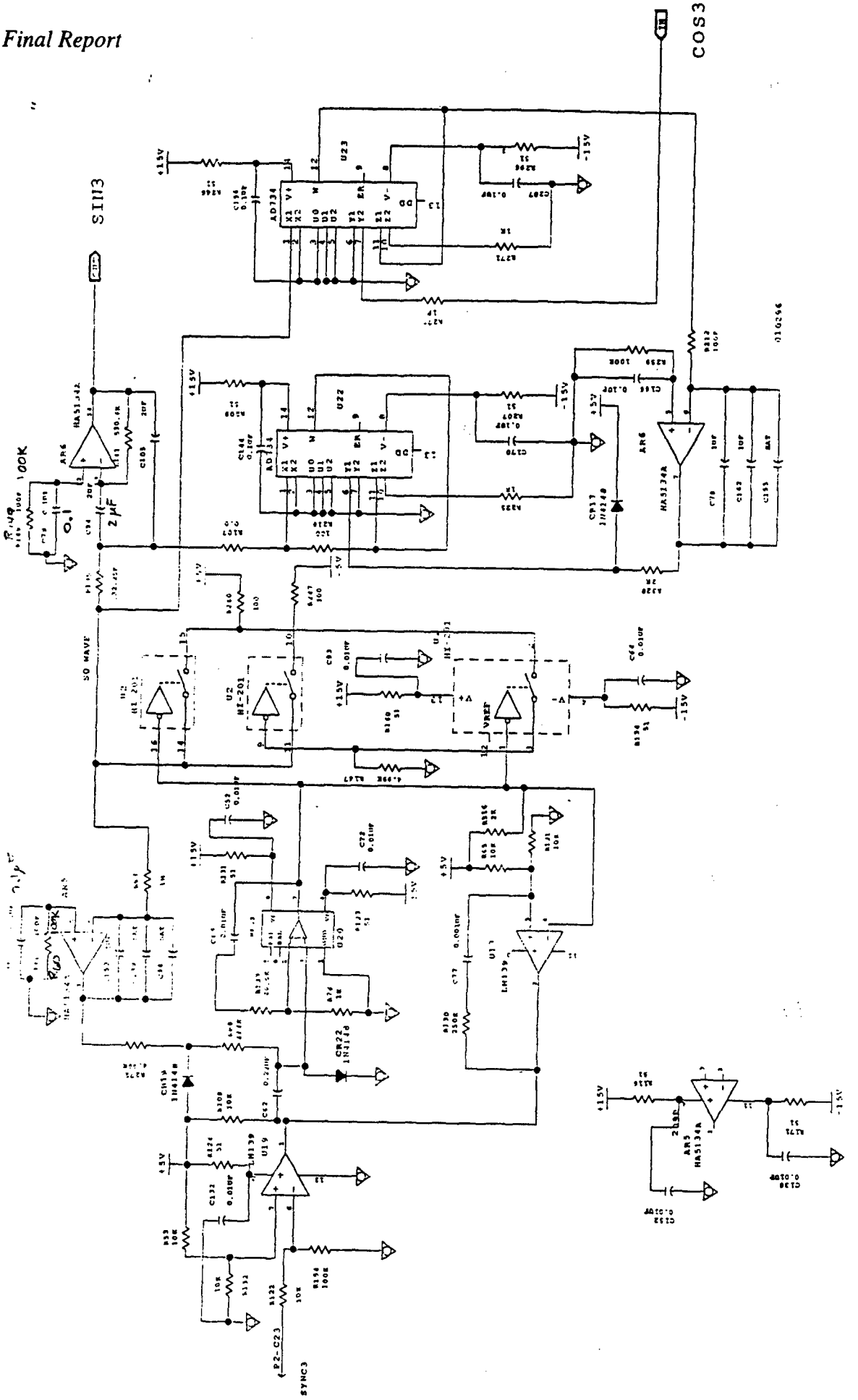

gain of 2
from W5,
1 from Y(k)
peak gain 3
ζ = 0.25
5 HZ
peak gain 3
ζ = 0.25
20 HZ
10K

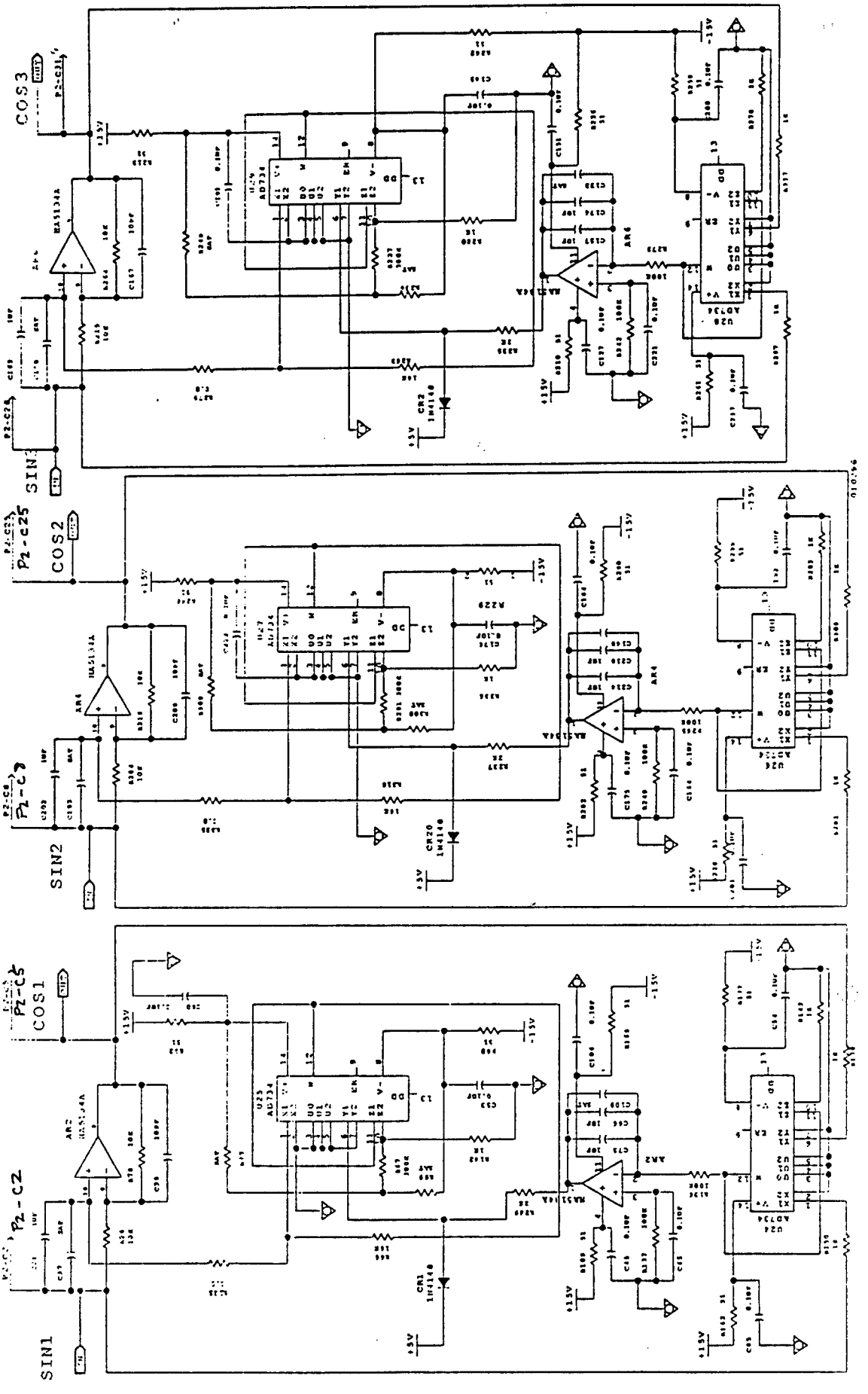

P1
P2
P3
U1
U2
U3
U4
U5
U6
U7
U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
AR1
AR2
AR3
AR4
AR5
S1
S2
S3
S4
S5
S6

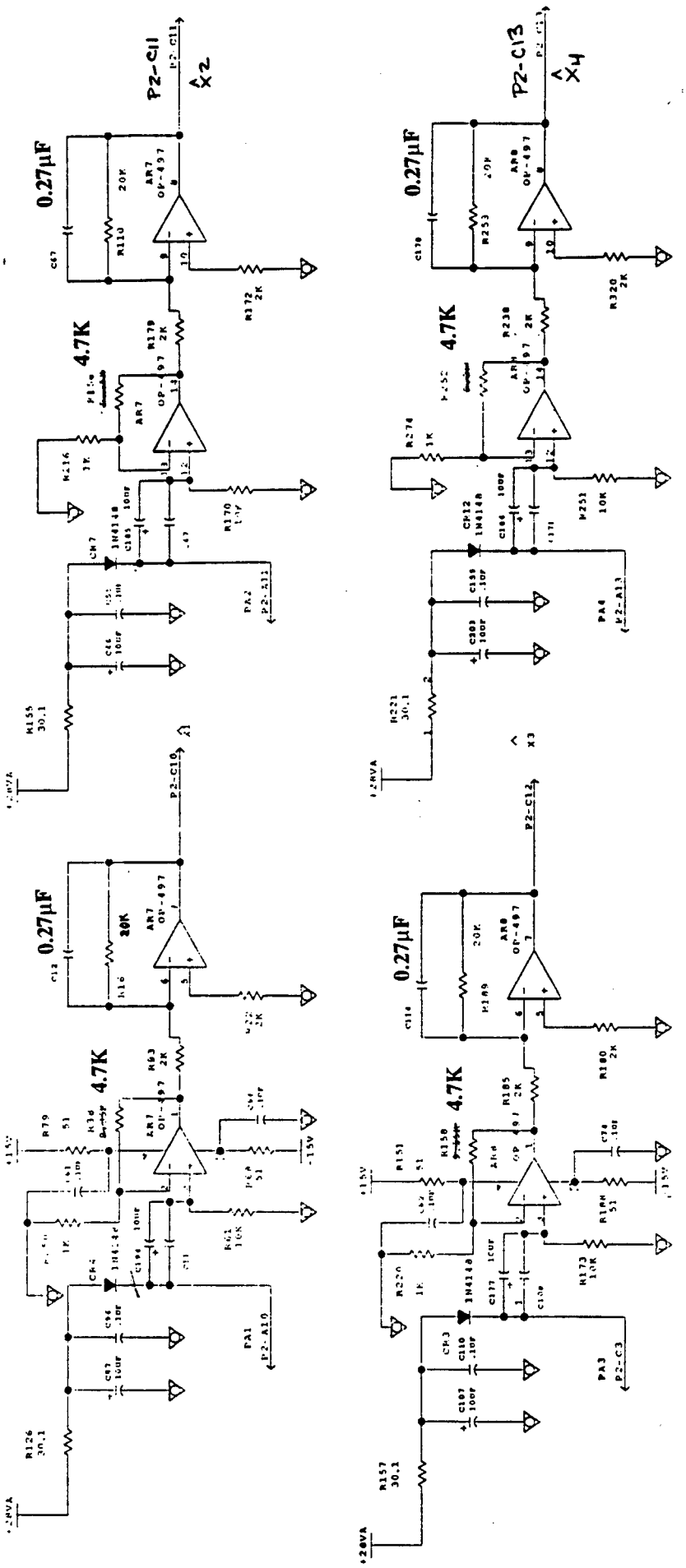
Sync Signal to Sine/Cosine Converter Cards

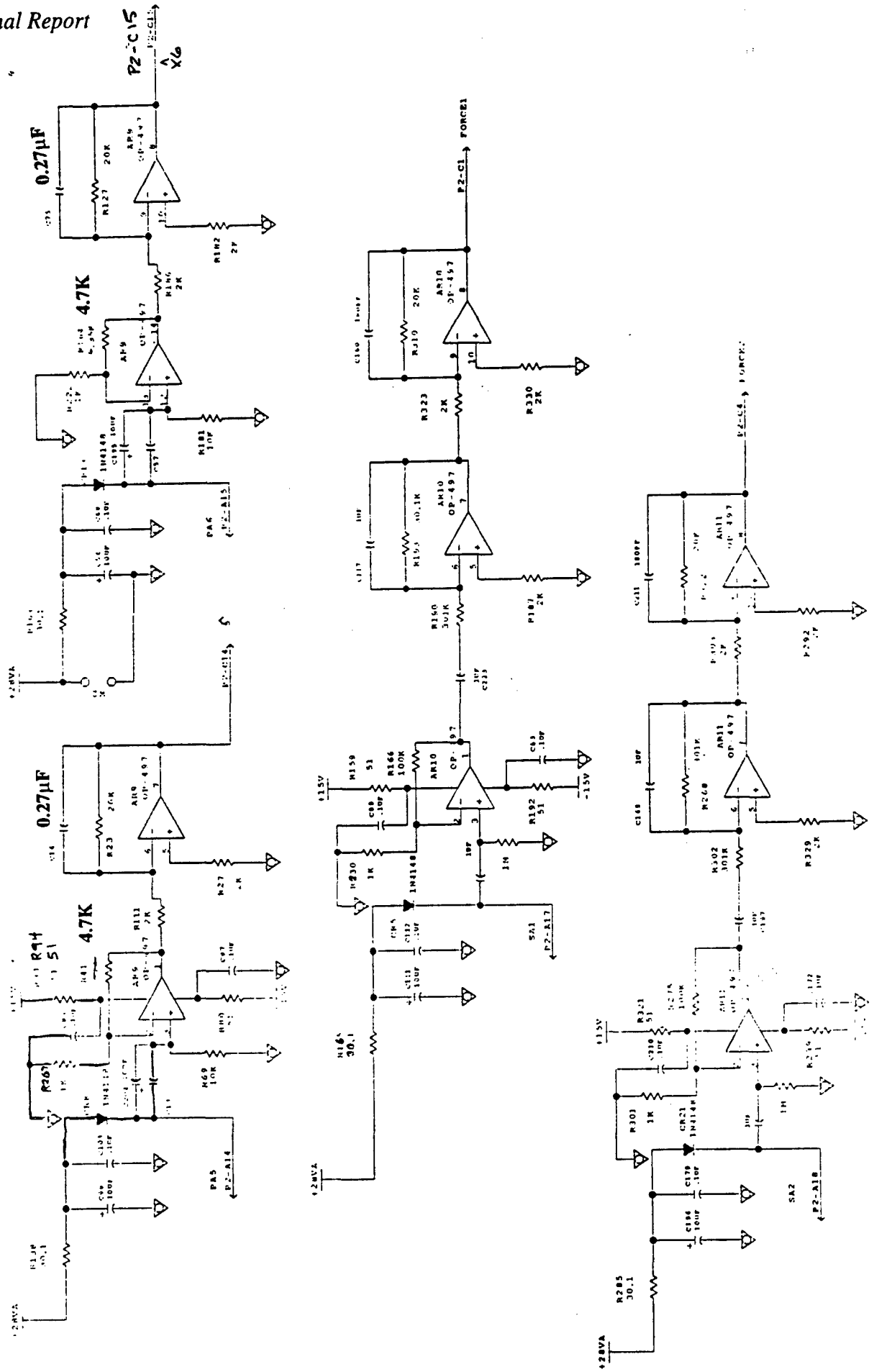

COS1
SIN1
R129
100K
0.1μF
SQ WAVE
SYNC1

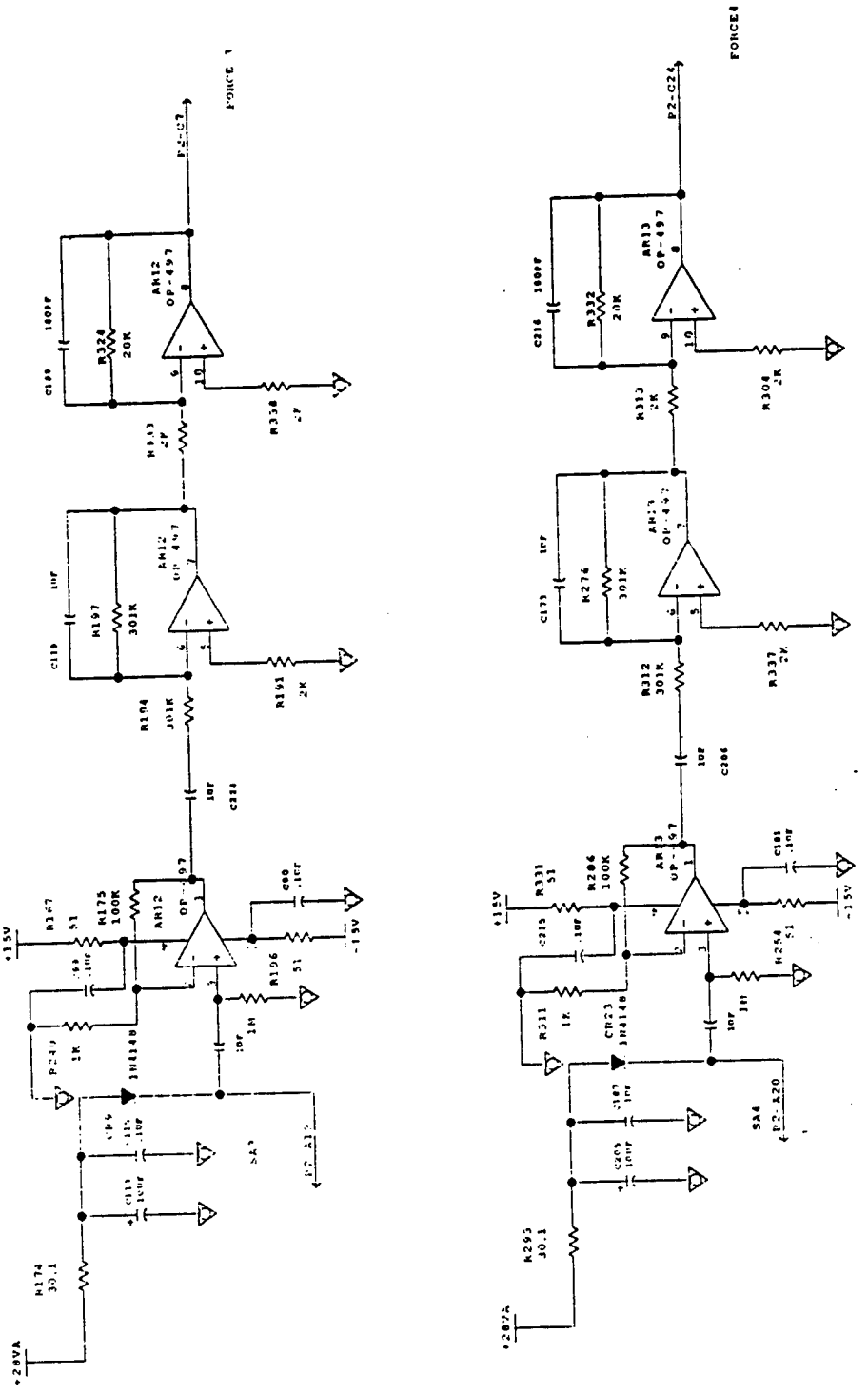

COS2
SIN2
SQ WAVE
VREF
SYNC2

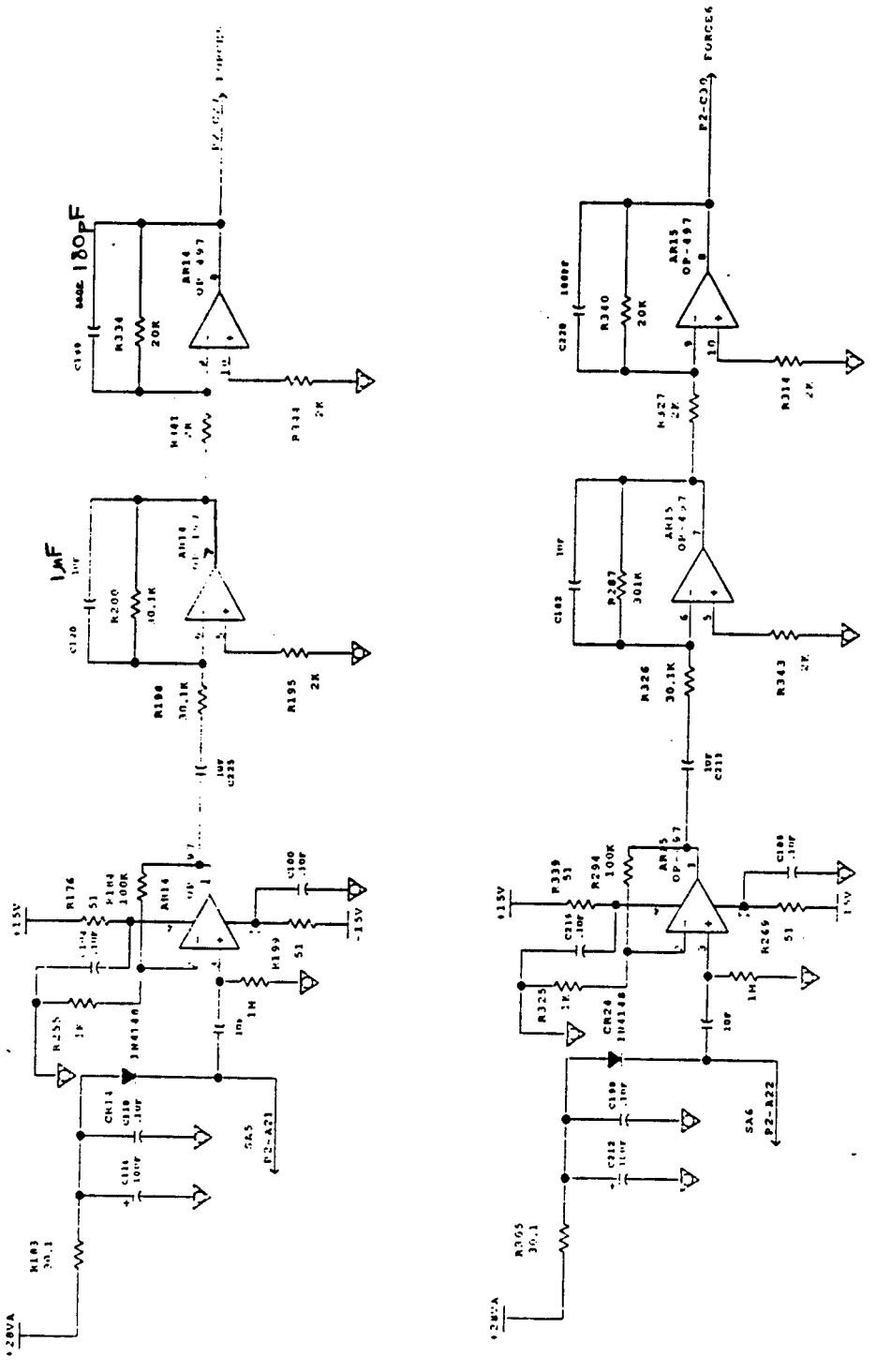

COS3
SIII3
SO WAVE
SYNC3

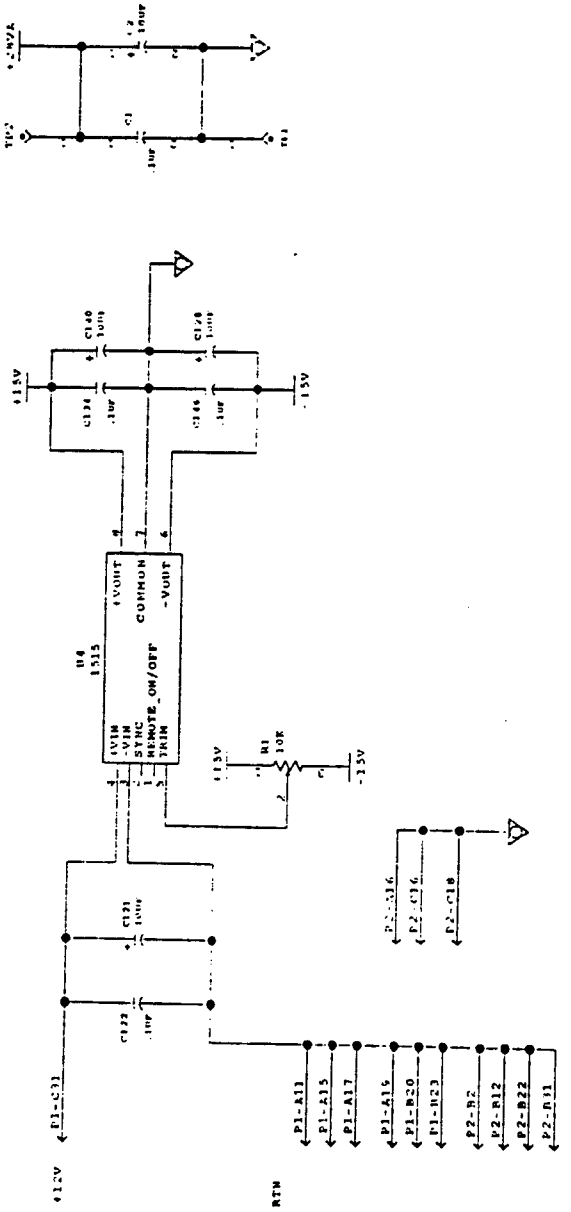

COS3
SIN3
COS2
SIN2
COS1
SIN1

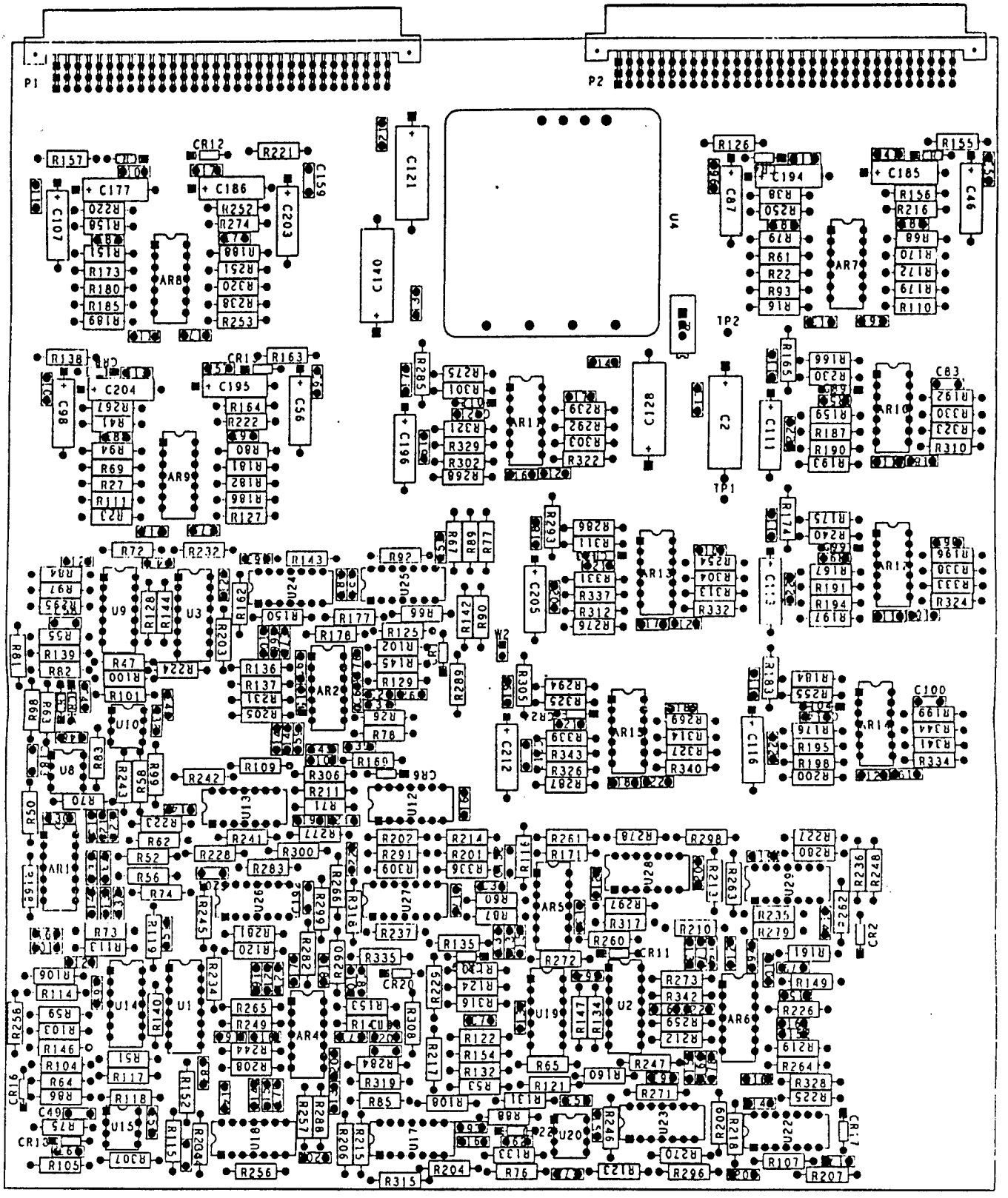

P2-C11
X2
0.27µF
4.7K
P2-C13
X4
0.27µF
4.7K
P2-C10
X1
0.27µF
4.7K
P2-C12
X3
0.27µF
4.7K


P2-C15
0.27µF
4.7K
R94
51
R69
10K
R265
P2-A14
P2-A15
P2-A17
P2-A18
FORCE1
R330
2K
R285
30.1
+28VA
-15V
+15V






+VOUT
COMMON
-VOUT
+VIN
-VIN
SYNC
REMOTE_ON/OFF
TRIM
+15V
-15V
+12V
RTN


P1
P2
U4
TP1
TP2
AR1
AR2
AR3
AR4
AR5
AR6
AR7
AR8
AR9
AR10
AR11
AR12
AR13
AR14
U1
U2
U3
U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
U20
U21
U22
U23
U24
U25
U26
U27
U28
CR2
CR6
CR11
CR12
CR13
CR16
CR17

DISTRIBUTION LIST

AUL/LSE Bldg 1405 - 600 Chennault Circle Maxwell AFB, AL 36112-6424	1 cy
DTIC/OCP 8725 John J. Kingman Rd, Suite 0944 Ft Belvoir, VA 22060-6218	2 cys
AFSAA/SAI 1580 Air Force Pentagon Washington, DC 20330-1580	1 cy
PL/SUL Kirtland AFB, NM 87117-5776	2 cys
PL/HO Kirtland AFB, NM 87117-5776	1 cy
Official Record Copy PL/VTS/Gary Yen Kirtland AFB, NM 87117-5776	2 cys
PL/VT Dr Wick Kirtland AFB, NM 87117-5776	1 cy